COURS PRATIQUE DE MICROPROCESSEUR AVEC LE MICROPROFESSOR MPF-IB

PREMIERE PARTIE

Concept de base

SOMMAIRE

I. INTRODUCTION

Nous avons tous eu l'occasion d'utiliser une machine à calculer électronique. Elle fait partie de la panoplie de l'étudiant d'aujourd'hui, et parfois même des lycéens, au détriment de la «règle à calcul».
A partir de son utilisation, nous allons établir son concept et ainsi aborder les micro-calculateurs, puis les micro-ordinateurs. Nous découvrirons ainsi ensemble le rôle essentiel que joue le microprocesseur dans ces systèmes.
Cette revue d'ensemble, nous permettra d'aborder plus efficacement l'étude des microprocesseurs et la microprogrammation, et ayant ainsi acquis suffisamment de connaissances, vous pourrez réaliser vos propres applications.
Dans l'introduction, nous ne nous intéresserons qu'aux calculatrices simples : type calculatrice de poche avec un affichage numérique et capables d'effectuer les quatre opérations arithmétiques (addition, soustraction, multiplication et division).
Au cours de cette présentation, nous découvrirons ensemble les principaux éléments qui constituent un micro-ordinateur.
Nous présenterons, plus en détail, le système complet MICROPROFESSOR MPF-1B qui constitue le support de notre étude.

I.1 Concept d'une calculatrice

Deux fonctions fondamentales apparaissent à l'utilisateur : le clavier et l'afficheur.
La première permet d'introduire dans la calculatrice les données à traiter ainsi que les opérations à réaliser.
L'affichage visualise les chiffres préalablement introduits ainsi que le résultat de l'opération.
Tant pour l'entrée que pour la sortie des informations, les symboles utilisés nous sont familliers. Les touches sont numérotées de 0 à 9, les signes arithmétiques sont «+», «−», «:» et «×» et les caractères de l'affichage, s'ils ne sont pas rigoureusement des caractères arabes, sont quasiment équivalents et se lisent sans ambiguïté.
Peut-on en déduire que le synoptique de la calculatrice de poche est conforme à celui de la figure 1 ?

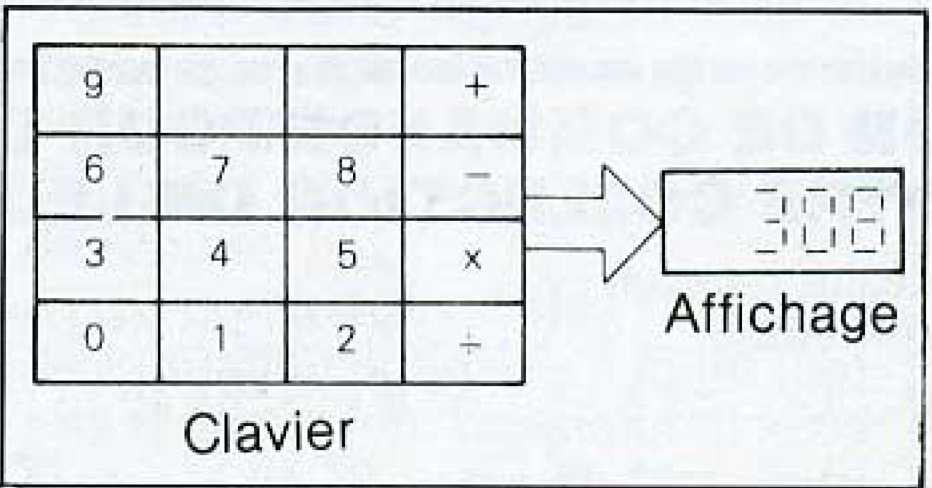
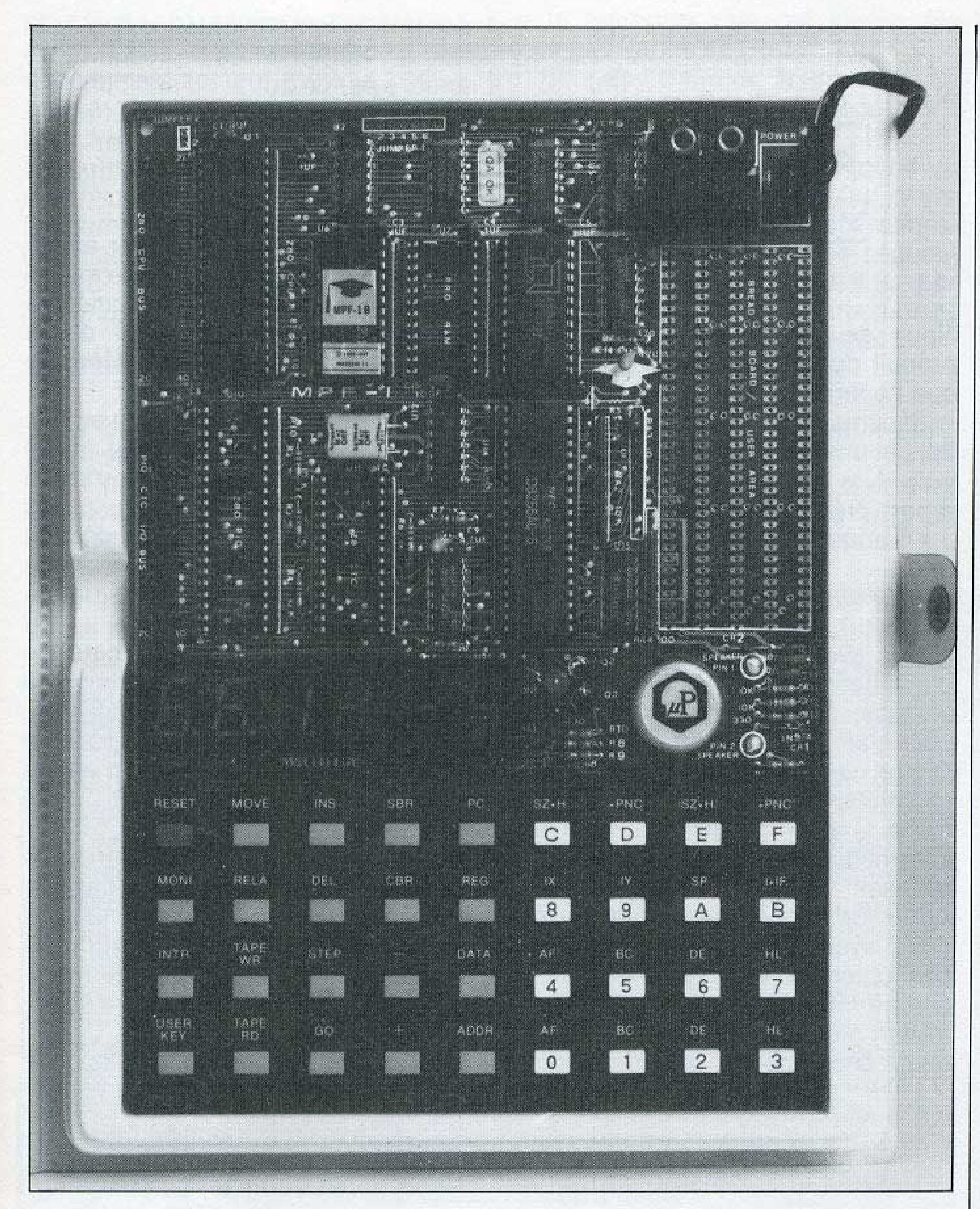


Fig. 1

Bien sûr que non !
On sent qu'il manque quelque chose entre **le clavier, organe d'entrée et**



l'affichage qui constitue **l'organe de sortie.**
Essayons d'en définir ses fonctions. En appuyant sur la touche 7, le symbole 7 apparaît à droite de la rangée d'affichage (figure 2).

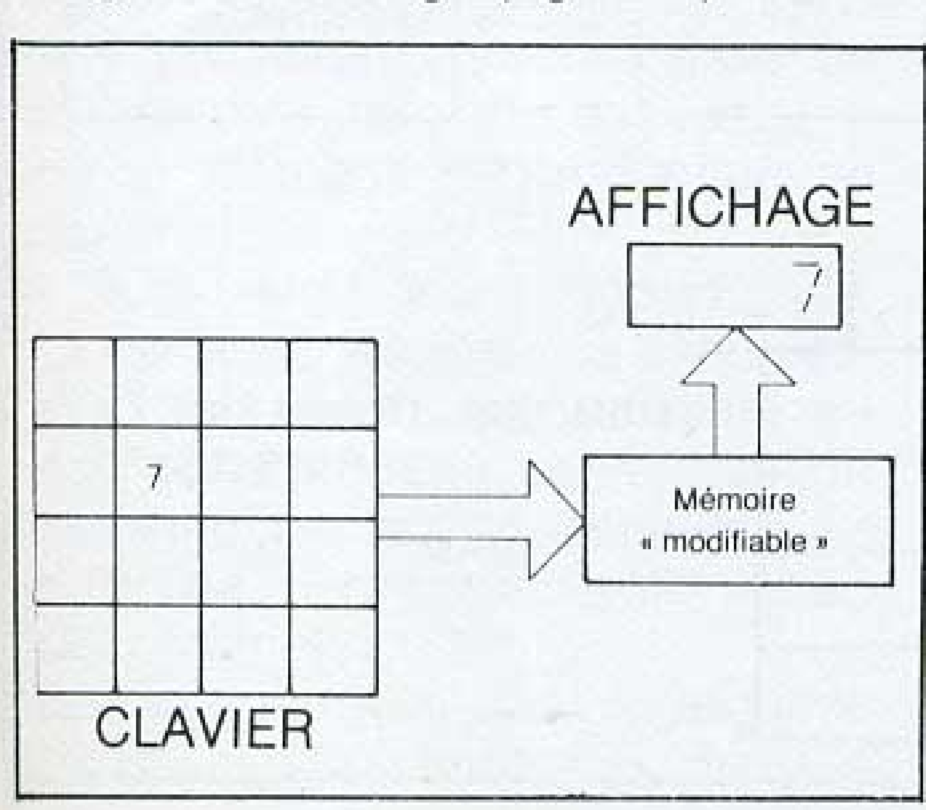


Fig. 2

Lâchons la touche. Le **symbole subsiste :** il existe donc entre le clavier et l'afficheur un **élément de mémoire.**
Quel type de mémoire ?
Nous aurions pu entrer le chiffre 3, ou tout autre nombre. Si nous appuyons sur la touche effacement, le chiffre disparaît. Donc, il s'agit d'une mémoire qui peut être écrite, effacée, puis écrite à nouveau : elle est **modifiable à volonté.**
Si après avoir introduit le nombre 7 on appuie sur la touche 3, le premier chiffre se déplace d'un emplacement vers la gauche pour occuper la position des dizaines tandis que la case à l'extrême droite contient le chiffre 3.
L'élément situé entre le clavier et l'afficheur dispose d'une taille mémoire suffisante pour garder plusieurs chiffres (au moins égale à la capacité de la calculatrice, 8 chiffres par exemple).
De plus, cet élément est doté d'une certaine logique, puisqu'il est capable d'identifier chaque touche enfoncée et de reconstituer le nombre en plaçant côte à côte chaque chiffre introduit individuellement. D'une manière plus générale, cet élément non seulement **mémorise les données** introduites par le clavier mais **effectue des opérations de traitement du nombre.**

Après avoir introduit le nombre 73, appuyons successivement sur les touches «×», «5» puis «=». Nous obtenons le produit de 73 par 5 soit 365.
Première constatation : contrairement aux données (7, 3, X, 5 et =) que nous avons dues introduire à l'aide des différentes touches correspondantes du clavier, ce mécanisme de la multiplication était déjà présent

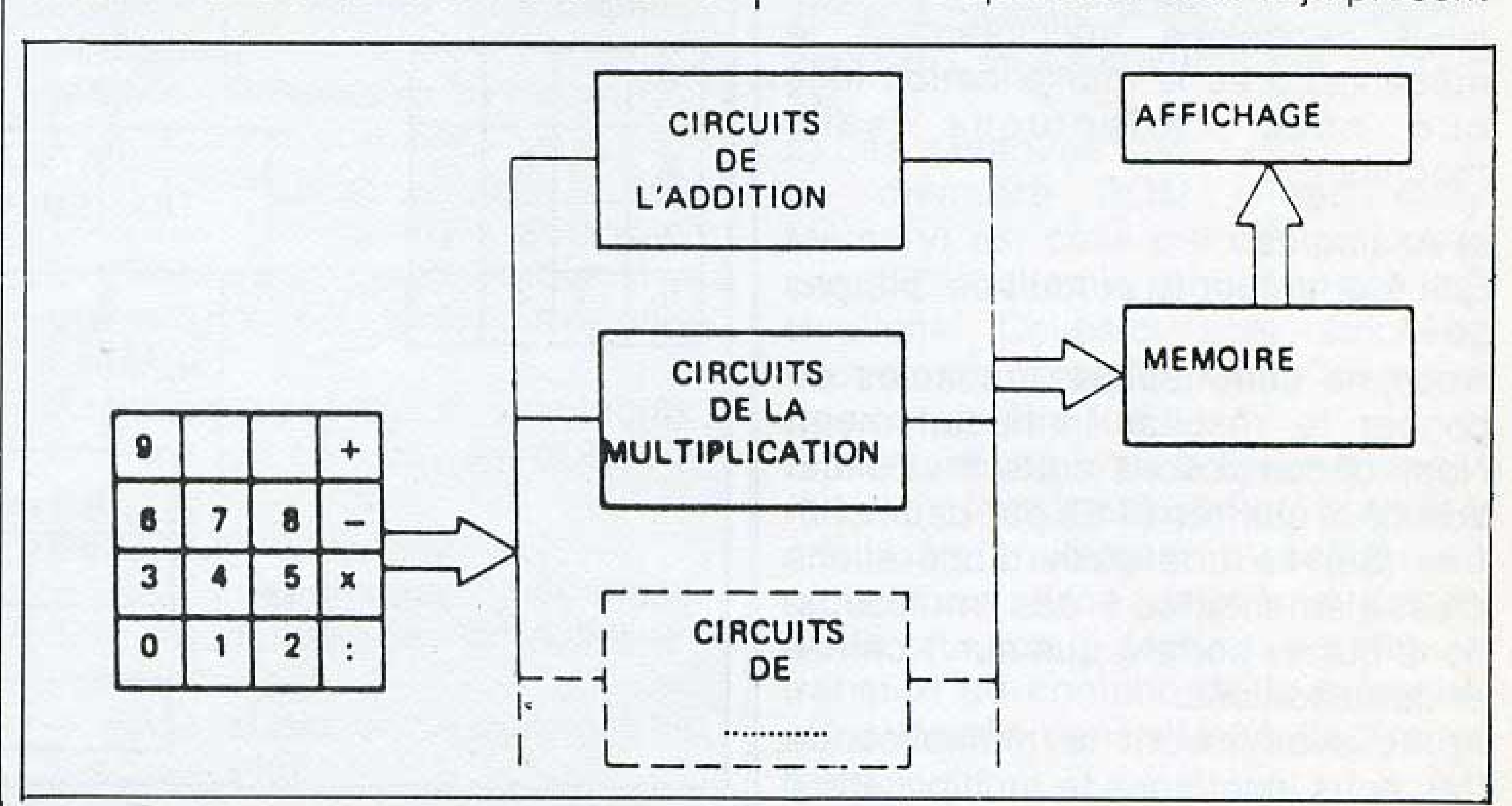


Fig. 3

dans la calculatrice. Il existe donc un circuit à la fois capable d'exécuter et de conserver en permanence le processus de cette opération. En appuyant sur la touche «x», nous avons sélectionné le circuit correpondant.

Comme nous pouvons réaliser les quatre opérations arithmétiques fondamentales, nous sommes tentés de penser qu'il existe quatre circuits distincts, un pour chaque fonction, ce qui nous conduit au synoptique de la figure 3.

Pour ajouter la fonction racine carrée d'un nombre, par exemple, il faudra ajouter le circuit correspondant à l'extraction d'une racine, et ainsi pour toute autre fonction supplémentaire souhaitée.
D'autre part, la multiplication fait appel à l'addition, au même titre que la division ne se conçoit pas sans la soustraction. De ce fait, la structure de la figure 3 est loin d'être satisfaisante, car trop rigide : elle conduit à augmenter les circuits spécifiques et entraîne une duplication des fonctions de base, telles que l'addition la soustraction, etc...
Intuitivement, on peut imaginer qu'un circuit plus flexible, de type universel, peut pallier à ces inconvénients. C'est ce que nous découvrirons avec l'Unité de Traitement.

I.2 L'Unité de Traitement

Le fonctionnement de l'Unité de Traitement est comparable au déroulement logique de la pensée chez l'homme lorsqu'il veut réaliser une opération arithmétique.
Nous rappelons succintement le mécanisme de la multiplication telle que nous l'effectuons (sans machine).

a) Analogies
Soit à effectuer le produit de 349 par 56
Nous ne sommes pas capables de donner le résultat immédiatement. Nous décomposons cette multiplication de 3 chiffres (349) par deux chiffres (56) en une suite d'opérations plus élémentaires : des multiplications qui ne portent que sur 1 chiffre et des additions.
Après avoir inscrit le multiplicande (M), nous inscrivons le multiplicateur (m) en dessous.

```
   3 4 9        (M)
 × 5 6          (m)
 ---------
  2 0 9 4
 1 7 4 5
 ---------
 1 9 5 4 4
```

Nous devons d'abord effectuer le produit de 349 par 6. Là aussi, bien que l'opération soit plus simple, nous la décomposons en une série de multiplications plus élémentaires : 9 x 6, puis 4 x 6 auxquelles on ajoute le report précédent, puis le produit de 3 x 6 auquel on ajoute le report antérieur.
Pour le chiffre 5 (unité de 10 du multiplicateur), nous procédons de même après avoir effectué un décalage d'une case à gauche, ce qui équivaut (dans le système décimal) à multiplier par 10 (la base du système).
Quand cette deuxième série de multiplications est terminée, nous effectuons la somme des deux résultats intermédiaires pour obtenir le résultat final.
Après ce rappel, analysons ce que nous avons fait. Nous ne pouvons pas obtenir d'emblée le produit de 349 par 56 (ou tout autre opération, à fortiori plus complexe), aussi l'avons nous décomposée en une succession (parfaitement ordonnée) d'opérations élémentaires : produit d'un nombre par un autre, ou addition de 2 nombres. Nous avons ainsi effectué environ 6 multiplications et une dizaine d'additions : soit un total de 16 opérations.

Dans une calculatrice (et par extension dans tout système à base microprocesseur), une opération quelconque est toujours le résultat **d'une succession d'opérations élémentaires.**
Prenons un autre exemple. Avec des briques, l'entrepreneur peut aussi bien construire un mur, une maison ou tout autre édifice plus ou moins important. Le résultat final ne dépend que de l'assemblage qu'il aura réalisé avec le matériau de base : la brique.
En micro-informatique, le matériau de base est l'INSTRUCTION ; elle représente une opération fondamentale qui elle-même correspond à l'aboutissement d'une suite d'opérations machine exécutées par l'Unité de Traitement.
Le PROGRAMME (la construction d'un édifice) est l'ensemble des instructions nécessaires pour mener à bien une tâche donnée. L'exécution du programme par le système représente, par analogie avec notre exemple, l'édifice.
Ainsi un problème complet, comme

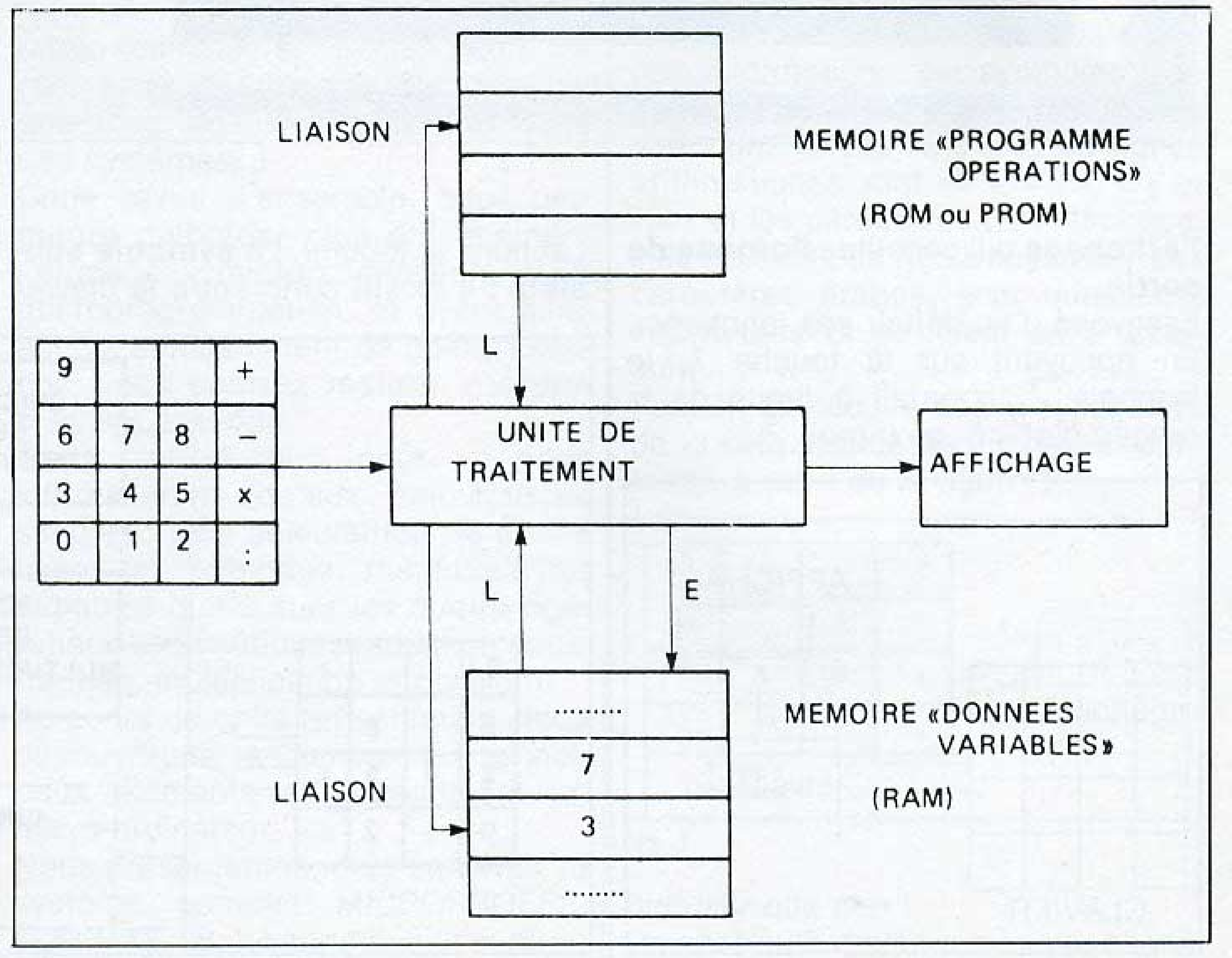


Fig. 4

pour les opérations arithmétiques, avant d'être résolu doit toujours être décomposé en une suite d'instructions.

b) Schéma fonctionnel

Nous pouvons élaborer un nouveau synoptique, tel qu'il est présenté par la figure 4, toujours dans le cas d'une calculatrice, dans lequel apparait l'Unité de Traitement, dont nous allons définir le rôle.

Cinq principaux éléments apparaissent :

— l'organe d'ENTREE ou périphérique d'entrée, le Clavier

— l'organe de SORTIE ou périphérique de sortie, l'Affichage

— l'organe de Traitement ou Unité de Traitement

— la mémoire morte qui contient le ou les programmes figés (ROM)

— la mémoire vive qui stocke les données variables (RAM)

c) Rôle de l'Unité de Traitement

L'Unité de Traitement exécute les INSTRUCTIONS.

A chaque instruction correspond un certain nombre de cycles au cours desquels une série d'opérations-machine sont réalisées. Ce séquencement est entièrement figé par construction et son bon déroulement réalise les fonctions contenues implicitement dans l'instruction.

Pour qu'il en soit ainsi, l'unité de traitement comporte deux modules distincts :

— L'Unité Arithmétique et Logique ou UAL (Arithmétic and Logic Unit en anglais ou ALU)

— l'Unité de Commande.

Le premier est, comme le nom l'indique, chargé de réaliser les **opérations arithmétiques** telles additions et soustractions, mais aussi les **opérations logiques** telles que ET, OU, OU EXCLUSIF, ainsi que les décalages à gauche ou à droite, les rotations, etc...

Le second, ou **Unité de commande** a pour but de **contrôler le séquencement,** c'est-à-dire préparer les différents circuits logiques et diriger les informations pour que chaque instruction soit correctement exécutée. Elle configure l'UAL dans la fonction requise : Addition ou Soustraction ou ET, etc...

Lorsque l'instruction en cours est achevée, c'est ce dernier module qui déclenche l'exécution de la suivante : il l'extrait du programme contenu dans la mémoire et configure l'Unité de Traitement en conséquence.

Ce qui diffère l'Unité de Traitement de la pensée humaine c'est la rapidité. Le temps d'exécution d'une instruction, c'est-à-dire d'un travail élémentaire est de l'ordre de quelques microsecondes.

Ce qui revient à dire qu'en 1 seconde, le micro-ordinateur peut accomplir quelques centaines de milliers d'opérations : on conçoit dès lors que le système soit «doué» d'une certaine puissance tout au moins de calcul : ce que nous savons déjà fort bien.

L'U.T. n'exécute qu'une seule instruction à la fois. La suite de ces opérations nécessaires pour réaliser une fonction donnée, constitue le programme. Dans celui-ci, chaque instruction est codée sous une forme susceptible d'être interprétée par l'Unité de Traitement et stockée dans une mémoire. Ceci implique que des liaisons existent entre les différents éléments et notamment les mémoires.

d) Les mémoires

La mémoire ROM (Read Only Memory) est celle qui contient pour chaque fonction la séquence des instructions. Celles-ci sont stockées sous forme de codes. Cette mémoire ne peut qu'être lue.

Sur le synoptique de la figure 4, apparait une première liaison qui permet à l'unité de traitement d'accéder au programme. Une deuxième liaison notée «L» pour lecture permet le transfert du contenu de la mémoire sélectionnée dans l'Unité de Traitement.

Les données (DATA) par contre sont

Unité de temps et ses «sous multiples».

— L'Unité de temps est la seconde : S

— Les sous multiples sont :

• 1 milliseconde (ms) qui représente 1 millième de seconde ou 10^{-3}s

• 1 microseconde (μs) qui représente 1 millionième de seconde ou 10^{-6}s

• 1 nanoseconde (ns) qui représente 1 milliardième de seconde ou 10^{-9}s

• 1 pico seconde (ps) qui représente 1 millième de nanoseconde ou 10^{-12}s

Mémoires

I. Mémoires «Mortes» ou «A lecture seule»

* Mémoire adressable en binaire qui contient des informations figées et permanentes, qui dans une utilisation courante dans un système ne peut être que lue.

Selon le mode d'introduction du contenu :

• ROM (Read Only Memory)

Le contenu, fourni par le client, est incorporé dans le composant au cours du procédé de fabrication (masque) par le constructeur.

• PROM (Programmable Read Only Memory)

Les informations sont introduites en une ou plusieurs fois dans le composant par l'utilisateur. L'écriture est une opération IRREVERSIBLE.

• EPROM (Erasable Programmable Read Only Memory)

Les informations sont introduites par l'utilisateur, mais à l'inverse des PROM'S, la mémoire peut être EFFACEE, pour permettre une nouvelle programmation.

Suivant le mode d'effacement :

U.V. PROM : Effacement par exposition aux rayons ultra violet (U.V.) pendant 15 à 30 minutes.

E.E. PROM : Effacement par une séquence électrique.

E.A. ROM : EPROM qui peut être électriquement programmée et électriquement effacée, sans être enlevée de son support du circuit.

II. Mémoires «vives» ou à Lecture et Ecriture

Mémoire adressable en binaire qui peut être lue et écrite. Toute nouvelle «ECRITURE» écrase la précédente. L'information est volatile : elle ne se conserve que si le composant est convenablement alimenté.

A la mise sous tension, avant toute écriture, ce contenu est quelconque.

Deux grandes familles :

— RAM dynamique : nécessite une séquence de rafraichissement

— RAM statique : aucun circuit supplémentaire.

mémorisées dans la «mémoire de données». C'est une mémoire modifiable ou RAM (Random Acces Memory ou «Mémoire à accès aléatoire») ce qui ne traduit pas effectivement le fait qu'elle puisse être lue et écrite, mais la terminologie est ainsi faite. Ce composant dispose d'un certain nombre d'emplacements ou cases mémoires dans lesquels l'Unité de Traitement vient déposer (écriture) ou rechercher (lecture) des données.
Comme pour la ROM, il existe une liaison de sélection.
En plus du canal «L» (lecture), constitué lui aussi de plusieurs fils, nous notons la présence d'un second canal, orienté en sens inverse (Unite de Traitement vers la mémoire) noté «E» pour l'écriture.
La RAM est une mémoire volatile d'une part et modifiable d'autre part, ce qui nous amène aux deux remarques suivantes :
a) Lors de la mise sous tension, le contenu d'une RAM est aléatoire (nous n'avons pas dit «0»). Par conséquent, toute opération de lecture sera toujours précédée d'au moins une phase d'écriture, pour obtenir quelque chose de cohérent.
b) Si l'utilisateur écrit successivement des données dans une même case mémoire, à la lecture, il n'obtient que la dernière donnée introduite. Toute nouvelle écriture dans une case mémoire de la RAM, écrase la précédente.

I.3 Micro-calculateur et micro-ordinateur

Une calculatrice est une machine capable d'effectuer des opérations arithmétiques sur des données numériques ou chiffres. C'est un micro-calculateur.
Un ordinateur, et ceci quelles que soient sa taille et sa dénomination (micro, mini, etc...) est un système capable non seulement de réaliser des opérations arithmétiques ou logiques, mais surtout de traiter des informations conformément à un programme, pourvu que les données lui parviennent sous une forme «assimilable».
La première différence qui apparait aussi entre l'ordinateur et la calculatrice est que le premier dispose d'une très grande puissance de traitement et d'adaptation à l'environnement.
Le second point est que l'ordinateur peut accomplir n'importe quelle tâche tandis que la calculatrice n'exécute qu'un nombre limité de fonctions qùi lui ont été assignées une fois pour toutes.

Pour réaliser une application, l'utilisateur élabore un programme en tenant compte de toutes les ressources du système. Ceci fait, et après vérification, il fige sur un support non volatile (PROM, ROM, etc...) les différentes opérations que l'ordinateur doit réaliser.
A ce stade là, la calculatrice et l'ordinateur (muni de son programme) sont opérationnels dès la «mise sous tension». La différence est que le programme de notre calculatrice a été réalisé par une tierce personne (sans que vous puissiez y modifier quoi que ce soit) tandis que le programme, que vous avez créé, (ou dont vous faites l'acquisition) vous en avez une totale maîtrise, soit pour le modifier soit pour l'adapter à votre application.
Ainsi calculatrice et micro-ordinateur possèdent une architecture matérielle générale très similaire, mais la première possède une configuration logicielle figée tandis que la seconde est entièrement sous le contrôle de son utilisateur.
Et cette différence est énorme.
Pour être parfaitement exact, un ordinateur dispose d'un programme minimum (ou résident), l'essentiel du logiciel doit être fourni par l'utilisateur sous une forme quelconque, (ROM, disquette, bande magnétique ou encore introduit à l'aide du clavier).
Le rôle du programme d'initialisation (Bootstrap en anglais) est précisément de permettre le chargement d'autres programmes, soit que l'utilisateur les élabore lui-même (introduction au clavier par exemple) soit qu'il en a fait l'acquisition (ROM ou disquette par exemple). Sans le programme résident, le système est totalement inopérant.
Si vous voulez «autopsier» votre calculatrice (ce qui est fortement déconseillé), vous risquez d'être déçus. Avec une forte probabilité vous n'y trouverez qu'un seul composant actif au lieu des trois annonces (en dehors du clavier et de l'affichage).
L'Unité de Traitement, la mémoire morte (ROM) et la mémoire vive (RAM) sont implantées sur une seule et même pastille de silicium (puce) et encapsulées dans un seul boîtier totalement hermétique (dans tous les sens du terme). La raison fondamentale est bien sûr d'ordre économique, et ceci est d'autant plus aisé à réaliser que les techniques de fabrication sont voisines.
La calculatrice électronique, connue de tous, nous a permis une première approche et surtout de bien comprendre le rôle des éléments fondamentaux qui constituent un microsystème. Elle se prête assez mal, maintenant pour poursuivre notre étude avec un maximum d'efficacité. Celle-ci sera désormais basée sur un matériel pédagogique, le MICRO-PROFESSOR MPF-1B, beaucoup mieux adapté et qui se prête parfaitement à l'enseignement de la microinformatique.

II. SYSTEME AVEC MICROPROCESSEUR

II.1 Définition d'un microprocesseur

Nous pouvons d'ores et déjà définir un microprocesseur comme étant un organe de traitement de l'information, constitué d'un ensemble d'opérateurs logiques et réalisé selon la technique des circuits intégrés.
Il possède deux modules essentiels. Le premier, l'Unité Arithmétique et Logique UAL (ALU en anglais) est un assemblage de fonctions logiques comme celles présentées dans les six premiers numéros de cette revue. C'est un circuit de type universel, très flexible, capable de réaliser l'addition, la soustraction, le ET logique, etc... Au cours d'un cycle d'exécution, il n'effectue bien sûr, qu'une seule opération, celle spécifiée dans l'instruction.
Le second module est l'unité de CONTROLE. L'exécution d'une instruction résulte de l'enchainement cohérent de différents cycles. Le nombre de cycles est déterminé par l'instruction elle-même. Le rôle de ce module est d'une part, de configurer correctement l'UAL et d'autre part, de générer les différents signaux de commande. Cette unité synchronise les diverses actions et veille au déroulement correct de chaque cycle. Cette partie fait essentiellement appel à la logique séquentielle (compteurs, registres, bascules, etc...) telle que nous l'avons décrite

dans les numéros 7 et 8 de LED MICRO.
Le déroulement du programme implique que les instructions stockées en mémoire parviennent à l'unité de traitement de manière séquencielle. Il en va de même pour les données à manipuler. Il faut donc réaliser une communication entre l'unité de traitement et les mémoires (ROM ou RAM). D'autre part, des échanges d'information peuvent s'effectuer avec l'environnement : saisie d'une touche enfoncée au clavier, affichage d'un résultat sur la visualisation, par exemple.
En partant de notre synoptique (simplifié), représenté par la figure 4, nous allons établir par étapes successives les caractéristiques des différentes liaisons nécessaires entre un microprocesseur et les organes extérieurs : les mémoires d'une part, les périphériques (clavier, affichage, etc...) d'autre part.
A l'issue de cette étude, nous obtiendrons l'architecture complète d'un système bâti autour d'un microprocesseur.

Quelques définitions

BIT : Le BIT représente un élément d'information qui peut prendre deux états et deux états seulement conventionnellement notés «0» et «1».
DIGIT : Le DIGIT représente un élément d'information numérique qui peut être binaire, décimal ou autre. Exemple le digit 7.
MOT : Le MOT est un regroupement de plusieurs bits dont le tout a une signification. Par analogie avec le MOT du langage courant qui est un ensemble de lettres ayant une signification. Exemple : MAISON, LIVRE, VOITURE, etc.
OCTET ou BYTE : L'OCTET (ou BYTE) est un ensemble de 8 bits. Généralement il constitue un MOT, mais ce n'est pas obligatoire.
QUARTET : Le QUARTET est un ensemble de 4 bits. Deux quartets constituent un OCTET. Un QUARTET représente un nombre décimal (0 à 9) exprimé en DCB (Décimal Codé Binaire).

II.2 Représentation de l'information

Etant donné que les opérateurs constituant un microprocesseur sont des fonctions logiques, il est tout naturel que l'information soit représentée sous forme binaire, tout au moins au niveau du composant.
Le traitement pourrait s'effectuer «bit à bit», d'une manière série, mais ce serait très préjudiciable du point de vue vitesse d'exécution et diminuerait les performances du système.
En pratique, les informations sont présentées sous forme d'un mot de plusieurs bits qui constitue le «format» standard d'un microprocesseur. Ce format est d'ailleurs l'une des caractéristiques essentielles des microprocesseurs et en constitue l'un des critères de classification par famille.
Actuellement, les microprocesseurs 8 bits sont les plus répandus. Ils représentent, depuis plusieurs années, un excellent compromis coût/performance. De plus, leur mise en œuvre, au sein d'un système complet, est relativement aisée.
Avant l'apparition des «8 bits», des quatre bits étaient disponibles sur le marché, mais ceux-ci ont tendance à s'évanouir. Par contre, avec les améliorations constantes des techniques d'intégration et une meilleure maîtrise des phénomènes physico-chimiques liés à la diffusion, la réalisation de circuits de plus en plus denses est possible. Les microprocesseurs 16 bits sont couramment fabriqués par plusieurs constructeurs (MOTOROLA, ZILOG, THOMSON, etc) et les «32 bits» commencent à faire leur apparition (MOTOROLA, NS, TEXAS,...). La complexité et la mise en œuvre de ces circuits sont telles que leur étude déborde largement du cadre de ce cours, nous ne les évoquons que pour votre information.
Notre cours portera sur le Z80[R] de ZILOG, qui possède une structure 8 bits, même si certaines opérations sont réalisables sur 16 bits, ce qui facilite son utilisation.

II.3 Les liaisons

Reprenons le synoptique de notre calculatrice (figure 5) qui constitue une version schématique d'un système et examinons les unes après les autres les liaisons qui existent entre les différents «blocs».
Au préalable, nous rappelons brièvement l'organisation d'une mémoire.
a) Mémoire
La mémoire est une unité capable de stocker des instructions et des données sous forme binaire. Elle est constituée d'un grand nombre de cellules élémentaires (plusieurs milliers), chacune pouvant contenir un seul bit (1 ou 0).

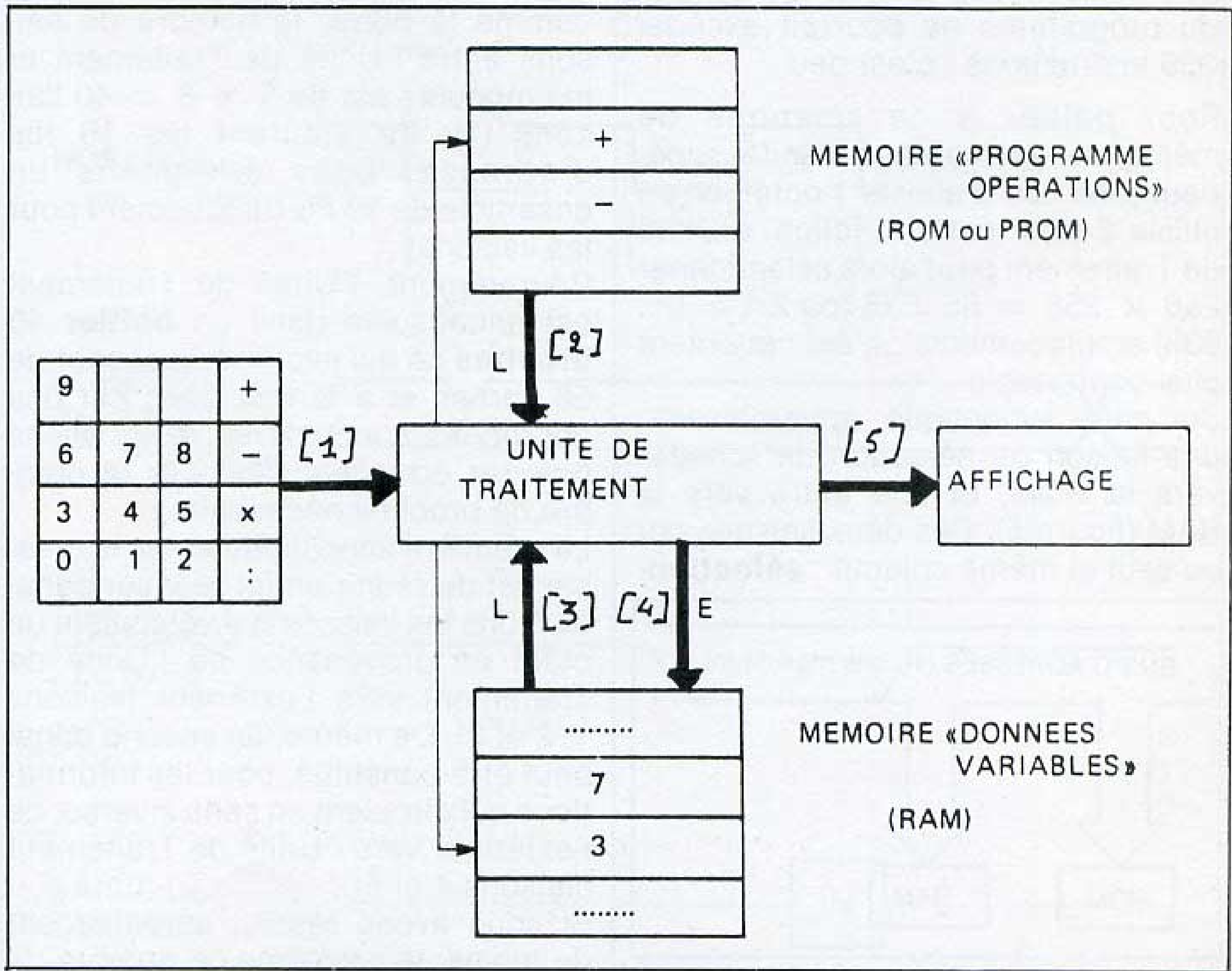


Fig. 5

Chaque emplacement de la mémoire est numéroté, il possède une adresse qui lui est propre de telle sorte que son contenu soit facilement localisable. (fig 6).

L'Unité de Traitement (et notamment l'UAL) effectue essentiellement des opérations sur des mots de huit bits ou octet : il est donc logique d'adopter une configuration identique, les échanges entre l'unité de traitement et la mémoire n'en seront que facilités. De sorte qu'à une adresse donnée N (figure 6) correspond non pas une seule case mais un ensemble de huit cellules contigües : un octet.

Adresse N − 1 →	1	0	0	1	0	1	1	0
Adresse N →	0	0	0	0	1	1	0	1
Adresse N + 1 →	1	1	1	0	1	1	0	0

Fig. 6

Certains composants mémoire sont à l'origine, organisés en mots de huit bits (2716, 2732, 2764 pour les ROM, et 6116 pour les RAM).

b) Les liaisons d'«ADRESSES»

Supposons que la liaison de sélection ne se fasse qu'avec 8 fils (de même taille que les mots lus), on ne pourrait sélectionner que 256 ($2^8 = 256$) cases mémoire, ou bien la capacité du programme ne pourrait excéder 256 instructions : c'est peu.

Pour pallier à ce «manque de mémoire», on saute à l'unité supérieure. Au lieu d'utiliser 1 octet, on en utilise 2 pour cette fonction. L'Unité de Traitement peut alors sélectionner 256 × 256 = 65. 536 (ou 2^{16} = 65. 536) emplacements : c'est nettement plus confortable.

Sur notre synoptique, apparaissent : une liaison de sélection de «cases» vers la ROM, et une autre vers la RAM (figure 5). Ces deux liaisons ont un seul et même objectif : **sélectionner 1 et seulement 1 seul emplacement.** Il est donc judicieux (et c'est d'ailleurs ce qui est fait) de confondre ces deux liaisons en une seule.

L'ensemble des fils de liaisons porte le nom de **Bus,** et comme il sélectionne l'adresse, c'est le **BUS D'ADRESSES (figure 7).**

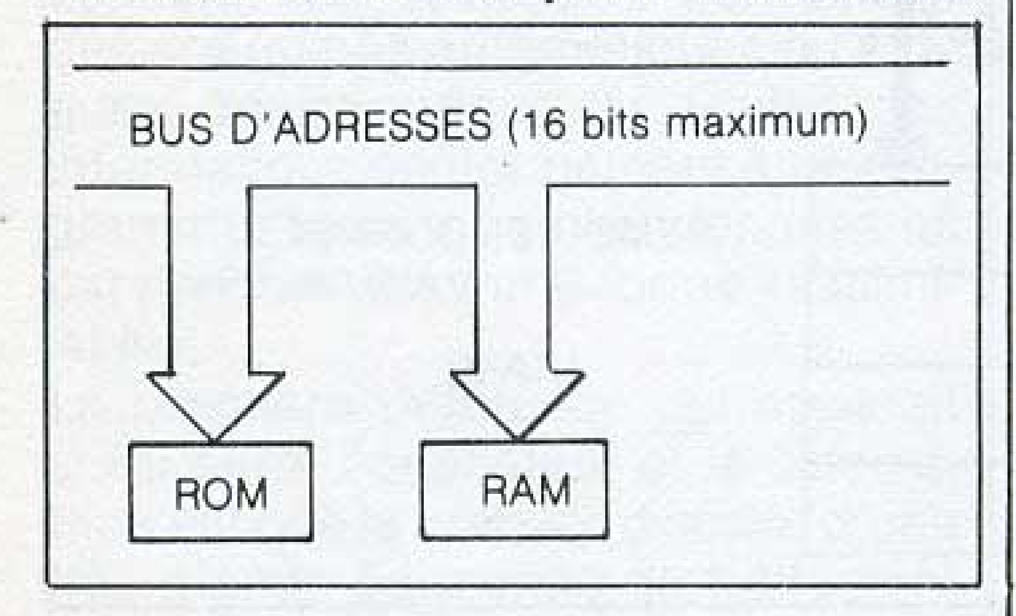


Fig. 7

b) Les liaisons de «données»

Sur le synoptique de la figure 5, cinq canaux différents apparaissent pour les «données». A savoir :

— liaison [1] clavier → Unité de Traitement

— liaison [2] ROM → Unite de Traitement

— liaison [3] RAM → Unite de Traitement

— liaison [4] Unité de Traitement → RAM

— liaison [5] Unité de Traitement → affichage

Chacun de ces modules fournit son information sous forme d'un mot de 8 bits pour rester en conformité avec le format. Dans un système simple comme le nôtre, le nombre de liaisons entre l'Unité de Traitement et les modules est de 5 × 8 = 40 liaisons (!). En ajoutant les 16 fils d'adresse, nous atteignons un ensemble de 56 fils (uniquement pour les liaisons).

Couramment, l'Unité de Traitement est encapsulée dans un **boîtier 40 broches** ce qui exclut de disposer de 56 sorties et à fortiori plus. Par des approches successives, nous allons montrer comment diminuer le nombre de broches nécessaires.

La première simplification envisageable est de réunir en un premier canal de 8 bits les liaisons qui véhiculent un octet en provenance de l'Unité de Traitement vers l'extérieur (liaisons 1, 2 et 3). De même, un second canal peut être constitué, pour les informations qui circulent en sens inverse, de l'extérieur vers l'Unité de Traitement (liaisons 4 et 5).

Si nous avons résolu, partiellement du moins, le problème (le nombre de fils est ainsi ramené de 40 à 2 × 8 = 16 !), nous en avons cependant créé un autre : plusieurs informations peuvent être présentées simultanément sur l'une des voies. Par exemple, une touche du clavier peut être enfoncée et envoyer son code à l'Unité de Traitement en même temps qu'une lecture de la RAM ou de la ROM s'effectue. Du côté de la sortie, le problème est identique.

Nous nous trouvons dans une situation identique à celle que connaît le professeur qui ayant posé une question à ses élèves reçoit en même temps la solution de plusieurs d'entre eux. Pour éviter cette confusion, **le professeur interroge l'un après l'autre chaque élève** ou uniquement ceux qui en font la demande. Ainsi, plutôt que chaque élève énonce sa solution simultanément dans la confusion générale, le professeur désigne un élève (et un seul) à la fois, et c'est lui seulement qui est autorisé à répondre.

Dans notre système, nous allons procéder d'une manière analogue. **L'Unité de Traitement** (qui joue le rôle du professeur) **sélectionne au moyen de 16 bits d'adresses, l'une des sources** (les élèves) **susceptible de lui envoyer une information,** c'est-à-dire 1 octet.

Cette source est soit un emplacement de la mémoire ROM ou RAM, ou le clavier. Etant bien entendu, **qu'à un instant donné, il ne peut y avoir qu'UNE SEULE source sélectionnée** et que c'est celle-là uniquement qui dépose sur la voie son information.

Ainsi, nous avons réduit à 2 canaux de 8 bits les liaisons de transfert d'information. Poursuivons sur notre lancée. Ne peut-on **confondre les deux canaux de 8 bits en un seul ?** Si, c'est ce qui est fait habituellement, et celui-ci est désigné par **«Bus de données».**

Cependant, en agissant de la sorte, il subsiste un problème. Certains modules extérieurs ne peuvent être que LUS (ROM, Clavier), si bien qu'une opération d'écriture est inopérante. D'autres ne peuvent être qu'ECRITS, comme le dispositif d'affichage. Par contre les éléments comme la RAM peuvent être soit LUS, soit ECRITS.

Comment un élément pourra-t-il identifier s'il doit fournir une information ou au contraire stocker une donnée ? L'Unité de Traitement, et plus précisément le module CONTROLE,

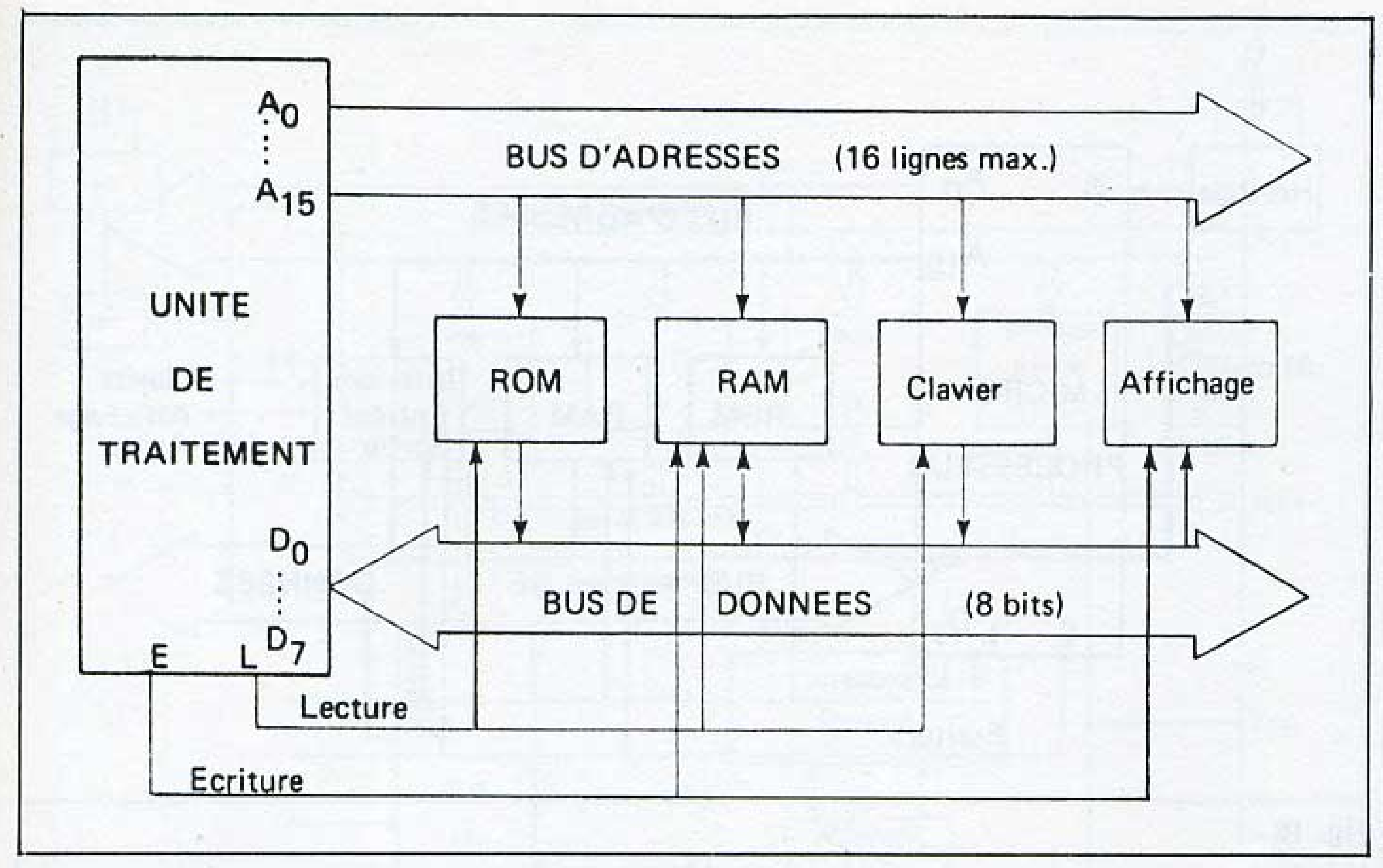


Fig. 8

génère deux signaux supplémentaires, l'un d'écriture, l'autre de lecture. Ces deux commandes, exclusives bien entendu, imposent un sens au bus de données (sortant ou entrant) et configurent le circuit périphérique concerné en Récepteur ou en Emetteur. Nous obtenons ainsi une nouvelle configuration (figure 8).

Sur laquelle apparaît :

— le BUS D'ADRESSES, composé de 16 lignes (au maximum). Les sorties sont habituellement notées A_0, A_1,... A_{15}

— le BUS DE DONNEES, bus bidirectionnel. Cette liaison de 8 bits envoie un message vers l'extérieur ou reçoit une information de l'un des éléments périphériques, les broches correspondantes sont, D_0, D_1... D_7.

— un troisième bus existe, c'est le bus de commandes. Pour l'instant nous n'avons identifié que deux commandes, l'une pour l'ECRITURE, l'autre pour la LECTURE.

II.4 Architecture d'un système

Le synoptique de la figure 8 nécessite encore quelques précisions pour constituer l'architecture d'un système.

a) Circuit d'horloge

Le microprocesseur travaille de manière séquentielle. Les instructions qui constituent le programme sont exécutées par l'Unite de Traitement les unes à la suite des autres, selon le déroulement fixé.

Chaque instruction, selon sa complexité, nécessite un certain nombre de cycles-machine. La durée de chaque cycle-machine est déterminée par le laps de temps qui s'écoule entre deux impulsions consécutives générées par un circuit extérieur appelé «HORLOGE».

Une période de l'horloge représente une unité de «Temps Elémentaire» ou T.E.

La durée d'exécution de chaque instruction est évaluée en T.E. Les plus courtes ne demandent que quatre cycles-machines, mais la plupart s'étendent sur des laps de temps plus longs.

Les fréquences d'horloge couramment utilisées sont de 1, 2,... 5 MHz, ce qui correspond à des Temps Elémentaires de 1, 0,5 et... 0,2 μs. Il est évident que plus la fréquence est élevée, plus le temps d'exécution est court. La fréquence limite de fonctionnement dépend essentiellement de la vitesse limite de chaque composant : le microprocesseur d'une part mais aussi les autres éléments notamment les mémoires (ROM et RAM).

Par exemple, avec une horloge de 2 MHz (T.E. = 0,5 μs), le temps minimum d'exécution d'une instruction (à 4 T.E.) est de 2 μs.

Quand une grande stabilité d'horloge n'est pas nécessaire, un circuit de type multivibrateur réalisé avec des composants passifs R-C constitue une solution économique.

Il est cependant préférable d'employer un oscillateur piloté par un quartz pour générer les impulsions d'horloge, la stabilité n'en sera que meilleure.

b) Cycle de lecture

L'exécution d'une instruction commence toujours par un cycle de lecture. Au cours de celui-ci, le contenu de la mémoire est amené dans l'unité de traitement.

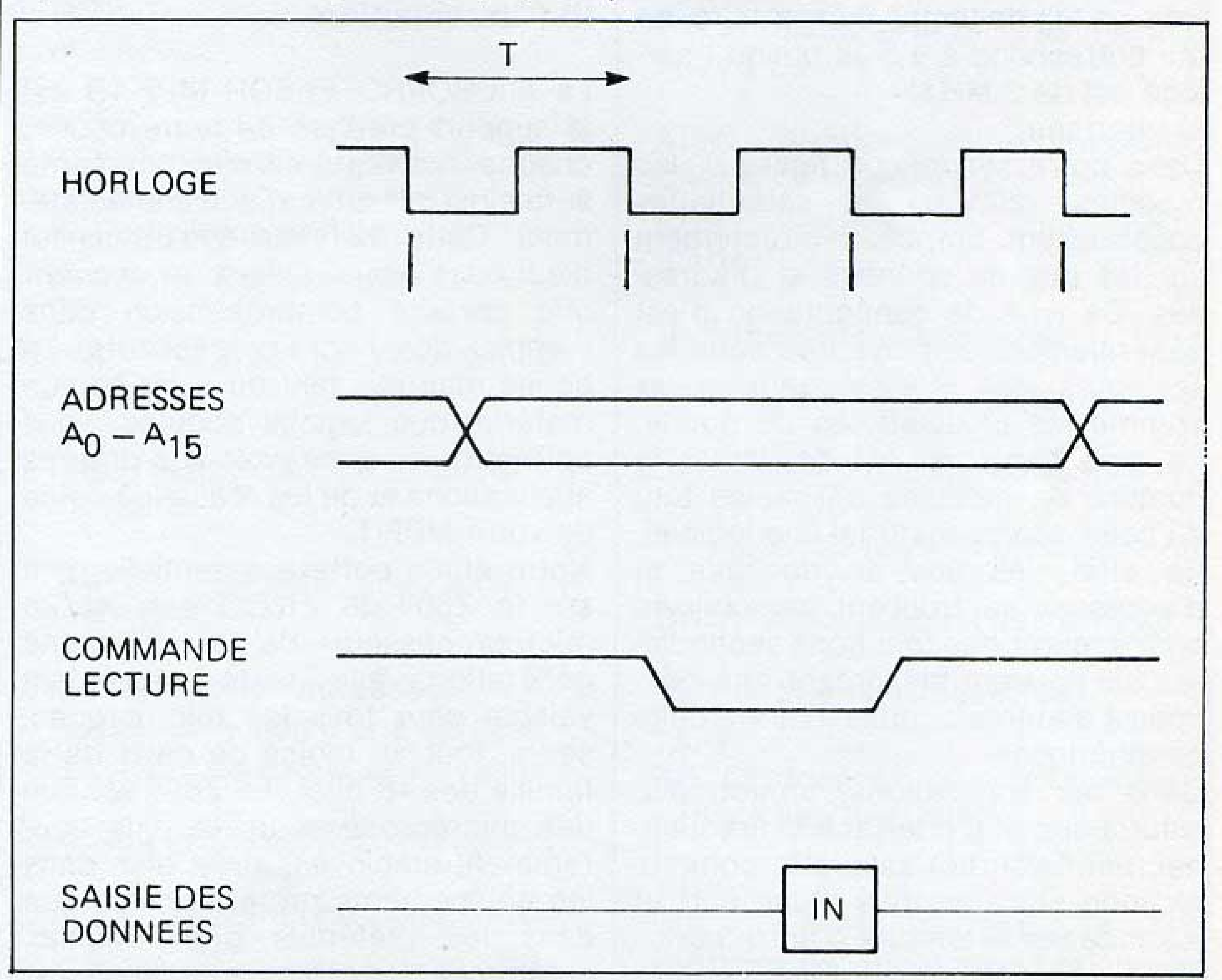


Fig. 9

Le code opération est placé dans le registre instruction qui appartient au module «Controle», décodé et exécuté.
La figure 9 montre le chronogramme d'un cycle de lecture.

L'Unité de Traitement fournit l'adresse de l'emplacement mémoire qui doit être lu. On dit que l'unité de traitement dépose une adresse sur le bus correspondant.
Le signal de lecture, généré par le module de contrôle, apparait ensuite. Le laps de temps (de l'ordre de quelques centaines de nanosecondes) entre l'instant de dépot de l'adresse et l'apparition du signal de lecture est nécessaire pour sélectionner la case mémoire ; il correpond au temps de propagation dans le décodeur notamment. Ce n'est qu'après un temps suffisant pour stabiliser l'adresse, que le signal de lecture apparait : celui-ci met en communication la case précédemment sélectionnée et le bus de données.
C'est au cours de la phase active du signal de lecture que l'Unité de Traitement échantillonne le bus de données : celles-ci seront «saisies» à l'intérieur d'une fenêtre étroite d'environ 500 ns (IN).
Le chronogramme de la figure 9 correspond à un cycle réel du Z80ᴿ. On note ainsi qu'un cycle de lecture dure trois unités de temps élémentaire, ce qui correspond à 1,5 µs quand l'horloge est de 2 MHz.
c) Interface
Dans notre synoptique figure 8, les modules «clavier» et «affichage» apparaissent branchés directement sur les bus de données et d'adresses. Ce type de configuration n'est généralement employé que dans les systèmes dits «très économiques» comme les calculatrices de poche. Le gros handicap est de limiter le nombre de modules utilisables tant du point de vue matériel que logiciel. En effet, les bus de données et d'adresses se trouvent immobilisés pour assurer des fonctions secondaires qui peuvent facilement être confiées à d'autres circuits, dits «circuits périphériques».
Dans les applications, un (ou plusieurs) circuit d'interface bidirectionnel (entrée/sortie) assure la communication entre les bus d'une part et les organes extérieurs d'autre part.
Nous obtenons ainsi l'architecture d'un système «micro-ordinateur» complet, représenté par la figure 10.

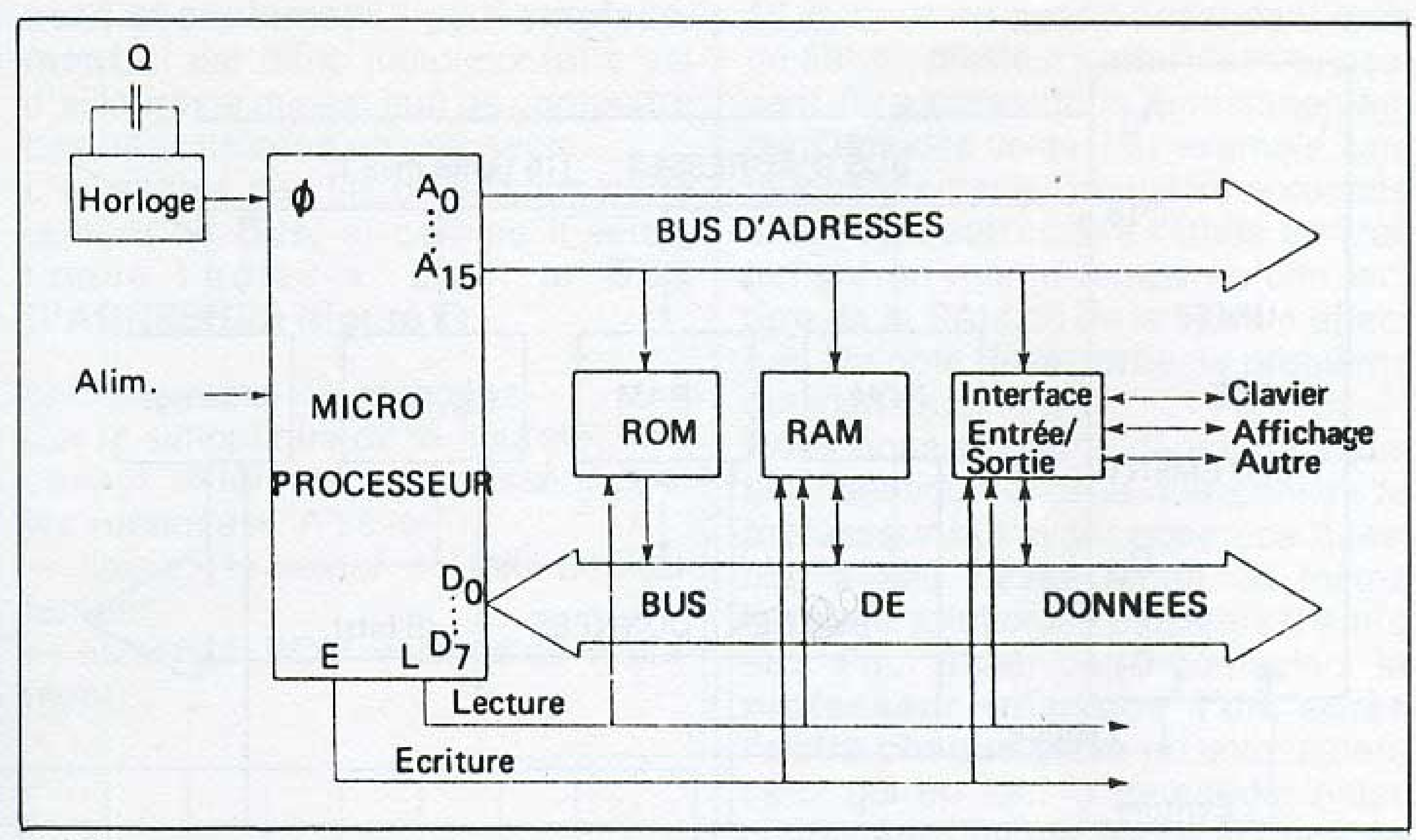


Fig. 10

Nous avons remplacé l'«Unité de Traitement» par le terme «MICROPROCESSEUR». Il apparait comme l'un des éléments qui constitue le système, le «centre nerveux» en quelque sorte.
Ainsi l'étude d'un système réel va pouvoir être abordée.

III. PRESENTATION DU SYSTEME MPF.1B

III.1 Introduction

Le MICROPROFESSOR MPF-1B est le support pratique de notre cours : chaque nouveau concept présenté sera ainsi mis en pratique immédiatement. Cette méthode dynamique et didactique vous aidera à acquérir une parfaite compréhension dans l'emploi des micro-processeurs. La bonne maitrîse, tant du point de vue matériel que logiciel acquise, vous permettra de concevoir vos propres applications et de les réaliser à l'aide de votre MPF-1.
Notre étude portera essentiellement sur le Z80ᴿ de ZILOG qui est un microprocesseur de la troisième génération, elle reste cependant valable pour tous les microprocesseurs, tout au moins de ceux de la famille des «8 bits». Le Z80ᴿ est l'un des microprocesseurs les plus couramment employés, aussi bien dans les équipements professionnels que dans les matériels grand public, notamment les ordinateurs familiaux. Un puissant jeu d'instructions (158) et de nombreux registres internes (22) confèrent à ce composant une grande souplesse d'emploi.
Le MPF-1B est un micro-ordinateur structuré autour du Z80ᴿ. Son concepteur a astucieusement simplifié cette unité centrale sans pour autant en limiter les performances. Il doit permettre à son utilisateur de s'autocontrôler et de suivre sa progression en micro-informatique. Dans ce domaine, plus encore que dans n'importe quel autre, seules théorie et pratique permettent réellement d'en «savoir plus».

III.2 Description

Le MPF-1B se présente sous forme d'un livre, comme l'indique la figure 11. Il est livré «prêt à l'emploi» avec son adaptateur secteur et une documentation complète.
Les composants sont montés sur un circuit imprimé, double faces, trous métallisés : ils sont visibles sans le moindre démontage. Chaque compo-

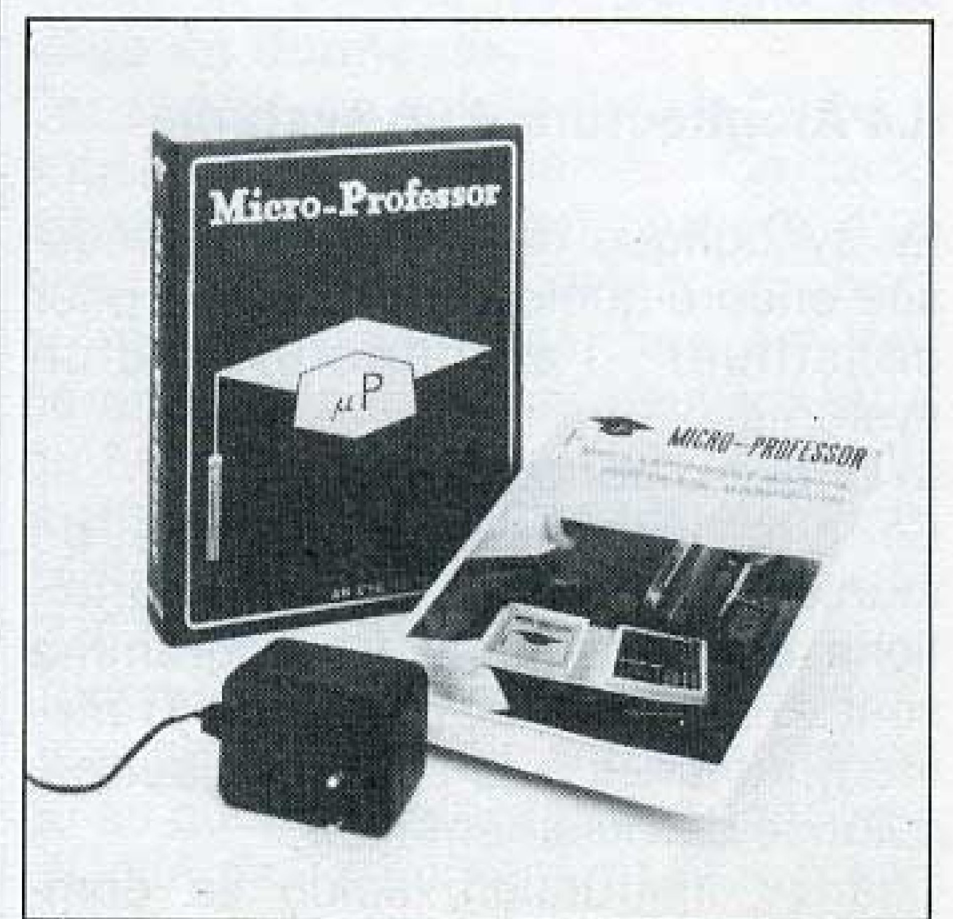


Fig. 11

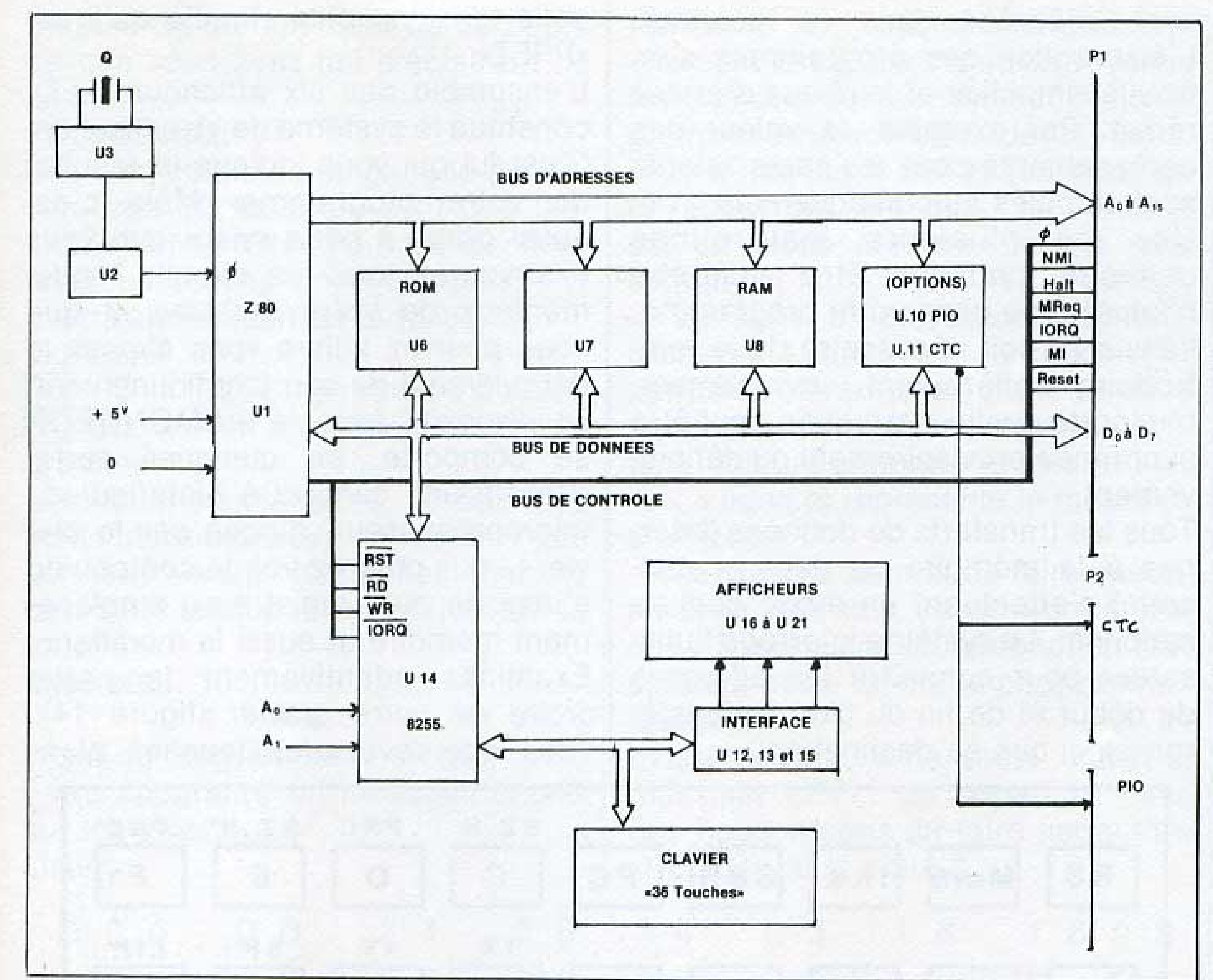


Fig. 12

sant porte sa référence et un repère figure sur le circuit : il sera très aisé d'identifier le circuit équivalent à chaque bloc et de faire une correlation entre synoptique et système réel.

L'architecture générale du MPF-1B est représentée par la figure 12, sur laquelle nous avons indiqué le repère des principaux composants pour faciliter leur localisation.

Jusqu'à présent, la mémoire ROM (ou EPROM) était réservée au programme tandis que la mémoire vive (RAM) contenait les données. Ceci est parfaitement exact dans une calculatrice, l'utilisateur n'ayant pas accès à la programmation. Par contre avec un micro-ordinateur, quel qu'il soit, le programme de l'utilisateur et les données coexistent dans la mémoire vive (tout au moins dans la phase étude). Il lui appartient de bien répartir chaque zone.

Les raisons sont très compréhensibles. Tout d'abord un programme, même simple ne se déroule pas souvent correctement à la première utilisation. D'autre part, dans une phase d'apprentissage, quand le concept étudié a été vérifié, il n'est pas nécessaire de le conserver.

Plutôt que de figer un programme dans une mémoire morte, qu'il faudrait ensuite effacer pour la modifier (si elle est effacable), l'utilisateur introduit son programme dans une zone de la mémoire vive et réserve une zone différente pour y placer les données.

De ce fait, les instructions du programme se modifient ou s'effacent aussi aisément que des «données».

Encore faut-il pouvoir entrer les instructions ou leurs codes dans la mémoire vive, au besoin les modifier et/ou les relire. Toutes les fonctions correspondant à ces commandes «utilitaires» sont figées dans une ROM (U6). Celle-ci contient le programme résident qui permet de rendre le système opérationnel.

III.3 Le moniteur

Le MONITEUR est l'ensemble des sous-programmes qui correspondent aux fonctions de base et aux commandes permettant l'exploitation aisée du système.

Parmi les fonctions, nous pouvons citer : l'initialisation à la mise sous

I. Fonctions générales

RS Remise à zéro du système

MONI Arrêt du programme et retour au MONITEUR

INTR Demande d'interruption masquable

USER
KEY Touche programmable

II. Fonctions «Data»

DATA Introduction d'une donnée

INS Insertion d'un octet en mémoire

DEL Suppression d'un octet en mémoire

MOVE Transfert d'un bloc de «données»

III. Fonctions «adresses»

ADDR Sélection d'une adresse

\+ Appel de l'adresse suivante

– Appel de l'adresse précédente

RELA Calcul d'une adresse relative

PC Rappel du compteur ordinal

IV. Fonctions «Exécution»

GO Exécute le programme à l'adresse spécifiée

STEP Exécute le programme en «pas à pas»

SBR Introduction d'un point d'arrêt

CBR Suppression d'un point d'arrêt

V. Fonctions «Registres»

REG Sélection d'un registre

\+ Appel du registre suivant

– Appel du registre précédent

DATA Introduction d'une donnée

VI. Fonctions «Cassettes»

TAPE
RD Chargement de la mémoire à partir d'une cassette

TAPE
WR Enregistrement de la mémoire dans une cassette

Tableau I

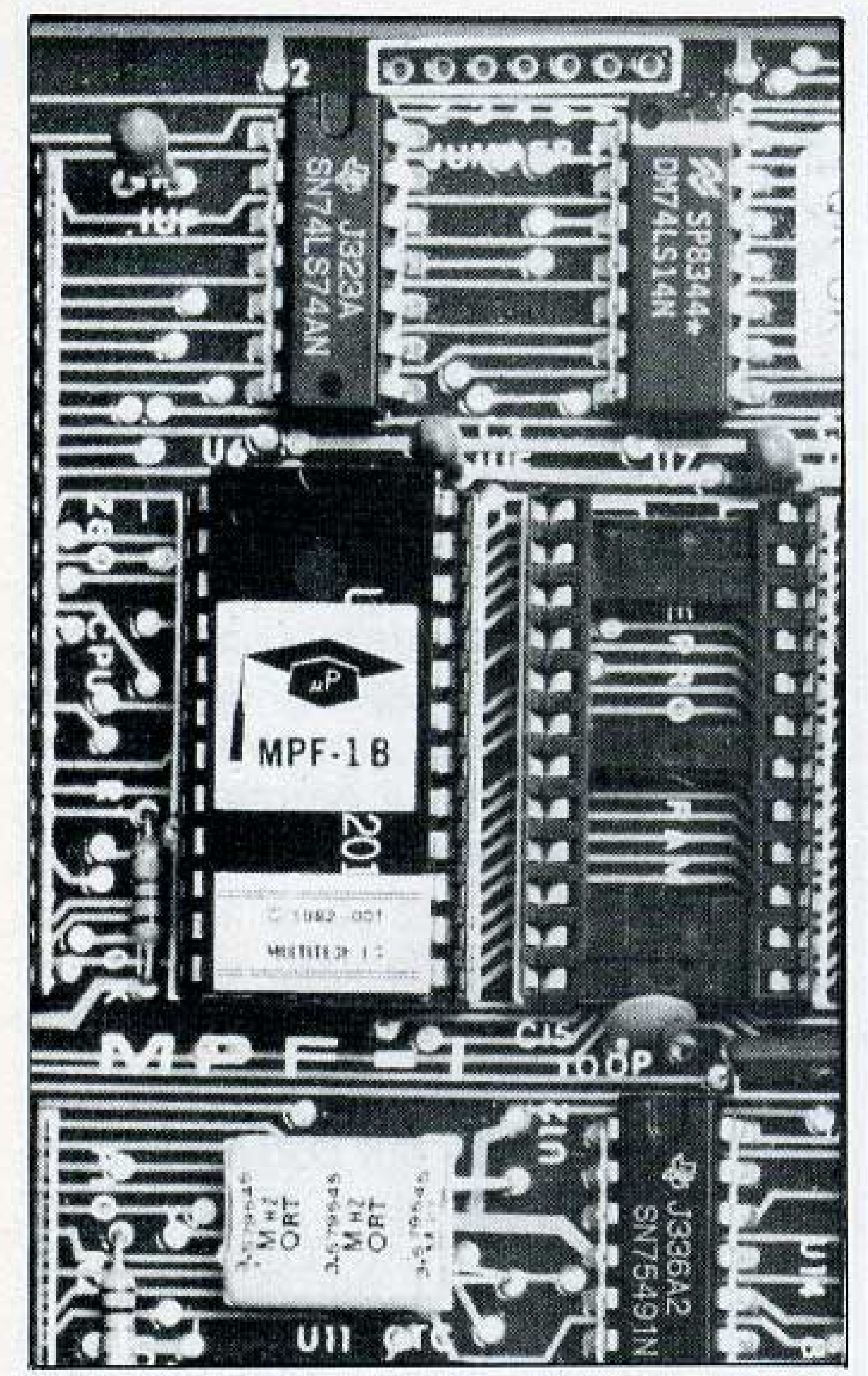


Fig. 13

tension, la gestion du clavier, le contrôle des afficheurs, etc... toutes sont accessibles mais non modifiables (puisque dans la ROM) par l'utilisateur.
Le tableau I résume l'ensemble des commandes.
La première fonction du Moniteur, est de permettre l'introduction des programmes dans la mémoire vive. Ensuite, l'utilisateur dispose de trois modes d'exécution : «intégral», «partiel jusqu'à un point d'arrêt fixé» ou en «pas à pas», c'est-à-dire instruction par instruction. Ce mode de fonctionnement est plus long, mais il permet de suivre le déroulement du programme et d'observer à l'issue de chaque phase le comportement interne du microprocesseur. Le contenu de tous les registres (au nombre de 22) ainsi que n'importe quel emplacement «mémoire» peut être visualisé et si besoin être modifié.
En phase d'apprentissage ou de mise au point de vos programmes, vous aurez recours au «pas à pas». Pour les programmes plus longs, vous laisserez le déroulement s'effectuer de lui-même jusqu'à un point repère que vous fixez. A partir de là, l'exécution peut se poursuivre dans n'importe lequel des trois modes. C'est l'équivalent du ralenti et de l'arrêt sur image, transposé à la micro-informatique.
Quelques fonctions «assembleur» sont intégrées dans le moniteur. L'élaboration des programmes s'en trouve simplifiée et le risque d'erreur réduit. Par exemple, la valeur des déplacements pour les sauts relatifs sont calculés automatiquement.
Une ou plusieurs instructions oubliées peuvent être inserées n'importe où dans votre programme, sans qu'il soit nécessaire de le reintroduire entièrement. Inversement, n'importe quelle instruction peut être supprimée provisoirement ou définitivement.
Tous les transferts de données (internes à la mémoire ou dans la cassette) s'effectuent en mode conversationnel. Le système interroge l'utilisateur pour connaître les adresses de début et de fin du bloc à transférer, ainsi que sa destination.

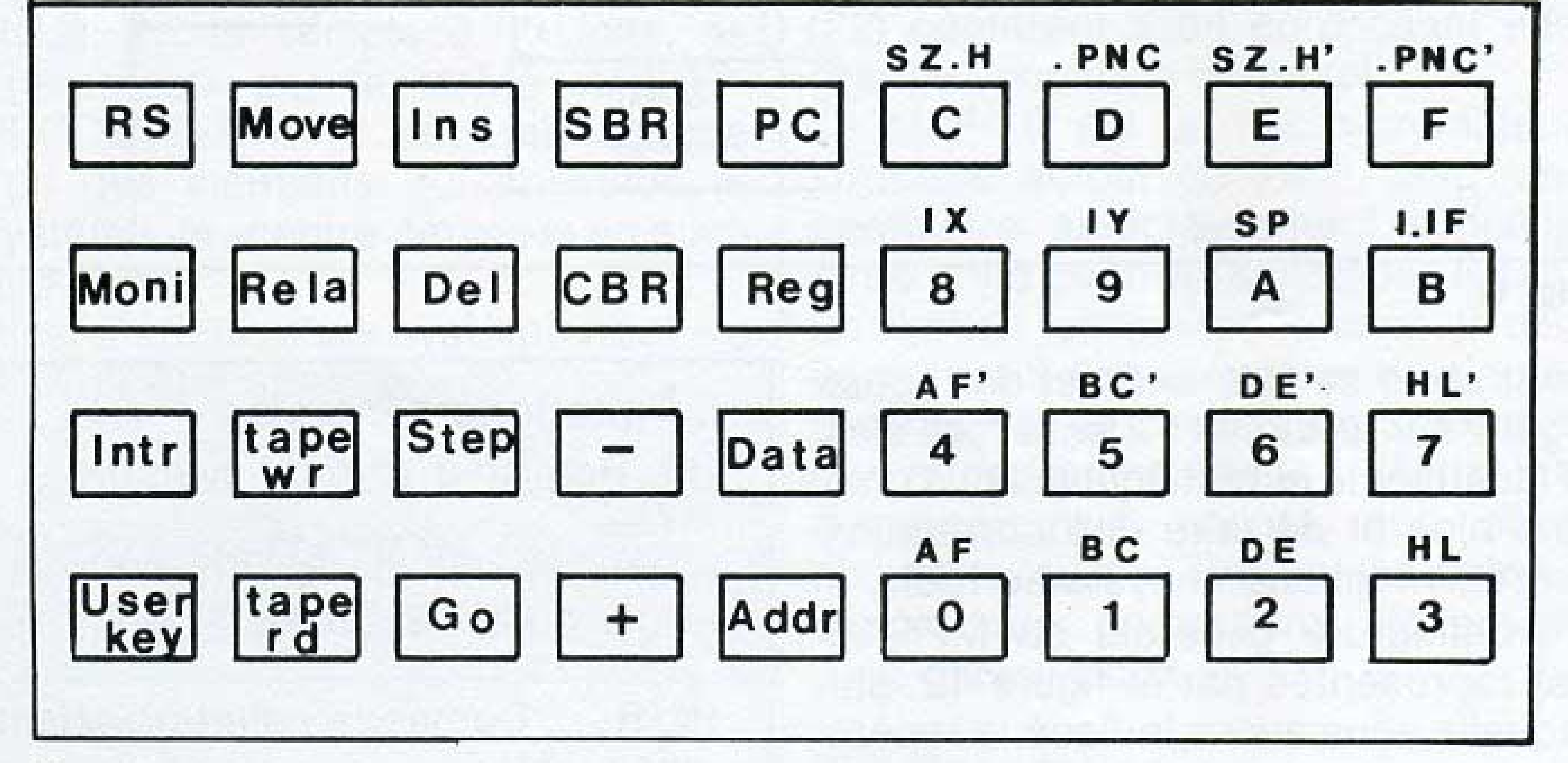


Fig. 14

L'ensemble des commandes du MONITEUR est décrit, exemple à l'appui, dans le Manuel Technique du MPF-1 qui accompagne le matériel.
En conclusion, le moniteur facilite l'exploitation du système, soulage l'utilisateur de tâches subalternes tout en augmentant leur fiabilité, celui-ci peut se consacrer à son application.

III.4 Application

Branchez votre MPF-1 avec l'adaptateur et... observez les afficheurs.
Presque immédiatement le message UPF-1 apparaît sur la visualisation en partant de la droite.
Débranchez, attendez quelques instants... et remettez sous tension ; la même séquence se déroule à nouveau. Le message que vous voyez ainsi apparaître est la phase finale d'un auto-test que votre micro-ordinateur effectue à chaque mise sous tension : cela signifie qu'il est «PRET».
L'ensemble des six afficheurs LED, constitue le système de visualisation. C'est lui qui vous indique le résultat de votre programme. Mais c'est aussi grâce à cette «visu» que vous examinerez tous les circuits fondamentaux de votre système et que vous pourrez suivre «pas à pas» le déroulement de son fonctionnement.
Le dispositif associé au MONITEUR se comporte, en quelques sorte, comme une caméra à l'intérieur du microprocesseur, dirigée par le clavier : vous pourrez voir le contenu de n'importe quel registre ou emplacement mémoire et aussi le modifier.
Examinez attentivement la partie droite de votre clavier (figure 14). Vous apercevez des touches blanches. Combien ? Observez l'inscription qu'elles portent, en commançant par 0, 1, 2...
Appuyez sur l'une d'entre elles.
Vous entendez un «son» bref et la led verte clignote. Ce sont deux autres moyens de communication avec votre micro-ordinateur. Ce message signifie que la touche enfoncée a été détectée, identifiée et pourtant elle n'apparaît pas nécessairement sur le système d'affichage.
C'est qu'avant d'introduire une information, il faut indiquer à votre micro-ordinateur ce qu'elle signifie et ce qu'il doit en faire : donc sélectionner l'une des fonctions du MONITEUR.
Pour vous familiariser avec votre matériel, vous allez maintenant introduire le programme qui suit et ensuite l'exécuter. Nous le commenterons ensemble.
Dans cette première partie, observez bien ce qui se passe, au besoin reprenez une manipulation... et au fur et à

mesure du déroulement de ce cours ce que vous avez fait s'éclairera de lui-même.

PROGRAMME

Les cinq colonnes de gauche du clavier, soit 20 touches, sont des touches «FONCTION». Pour les identifier nous utiliserons un rectangle [   ] avec l'abréviation de la fonction.

Exemple

[RS] RS = RESET ou remise à zéro

[GO] GO = Aller à ou Exécuter le programme

Les touches blanches, lorsqu'elles sont utilisées pour introduire une donnée seront désignées par le caractère unique tel qu'il figure sur la touche

Exemple :

4, 7, F etc...

Objet du programme :

Faire apparaitre un message donné sur les afficheurs.

Listing :

```
1800  DD  21  08  18   Ld  IX , 1808
  04  CD  FE  05       CALL  05 FE
  07  76               HALT
  08  00  BD  85  85
  0C  8F  37
```

Manipulation (détaillée pour la première)

Tout en observant bien ce qui s'affiche, appuyez dans l'ordre :

```
a)  [RS]  [PC]
b)  DD  [+]  21  [+]  08  [+]  18  [+]
    CD  [+]  FE  [+]  05  [+]  76  [+]
c)  00  [+]  BD  [+]  85  [+]  85  [+]
    8F  [+]  37
d)  [RS]  [PC]  ...  [GO]
```

Observez !

Le message affiché est constitué des 6 codes (00, DB, 85,... etc) de la ligne C. A l'aide de l'annexe 3 (page 94 du manuel technique du MPF-1), identifiez les codes des caractères.

Pour voir par vous-même si vous avez bien compris, faites apparaître le message de votre choix, en modifiant la ligne C. Notez que l'ordre dans lequel le message est introduit est inversé. Les codes sont placés de gauche à droite : blanc (00), O (BD), L (85), L (85), E (8F) et H (37). Reprenez le programme à partir de a).

Commentaires :

— La ligne a) permet d'initialiser le système. La touche PC après une remise à zéro, positionne automatiquement le pointeur d'adresse (nous reviendrons sur cet élément) sur le premier emplacement de la RAM, c'est-à-dire en 1 800 H (adresse en héxadécimal). De plus, le système est en mode écriture, ce qui rend l'appui sur la touche DATA facultatif.

— La ligne b) correspond au programme. L'appui sur la touche [+] permet d'incrémenter d'une unité le pointeur d'adresse. Le système restant toujours en mode «écriture».

— La ligne c) représente le message à afficher

— La ligne d) permet de faire exécuter le programme. Notez que la touche PC est suivie d'une fonction GO, qui permet de lancer le programme.

Question :

Existe-t-il une différence entre le message UPF-1 du départ et celui que nous venons de faire apparaître «HELLO» ou le vôtre ?

Le premier UPF-1 est APPARU AUTOMATIQUEMENT sans aucune intervention de votre part, si ce n'est la mise en marche. Par contre, nous avons du introduire le second message au moyen du clavier. Si vous débranchez le système, tout disparaît et il faut recommencer. Inversement, nous pouvons afficher, dans le second cas, le message de notre choix, ce qui était impossible dans l'autre cas.

Le premier message est inscrit par le constructeur dans la «mémoire morte» ou ROM. C'est l'aboutissement d'un auto-test qui démarre automatiquement à la mise sous tension. Donc, ce message ne peut être modifié.

Par contre, notre programme est inscrit dans la «mémoire vive» ou RAM. Sans difficulté et avec un succès, vous avez pu inscrire d'autres codes et ainsi obtenir un autre message. La nouvelle écriture écrase la précédente.

Si vous débranchez votre MPF-1 et qu'une fois remis sous tension vous exécutez directement la ligne d, vous n'obtenez aucun message.

Tout ceci confirme ce que nous avons dit plus avant :

— La ROM est non volatile. Son contenu ne disparait pas quand elle n'est plus alimentée. Elle peut être lue mais pas écrite, le message UPF-1 apparaît automatiquement.

— La RAM est volatile. Son contenu disparaît quand elle n'est plus alimentée. Après une coupure de l'alimentation du système, notre programme, contenu dans la RAM a disparu. Elle peut être lue, mais aussi écrite. Toute nouvelle écriture écrase la précédente.

III.5 Affichage

Chacun peut se demander comment on peut établir une correspondance entre

le code 37 et le graphisme H
le code 8F et le graphisme E
le code 85 et le graphisme L
etc...

Notre dispositif d'affichage est constitué de 6 afficheurs, chacun fait de 7 segments (fig 15) et d'un point décimal : soit huit éléments au total.

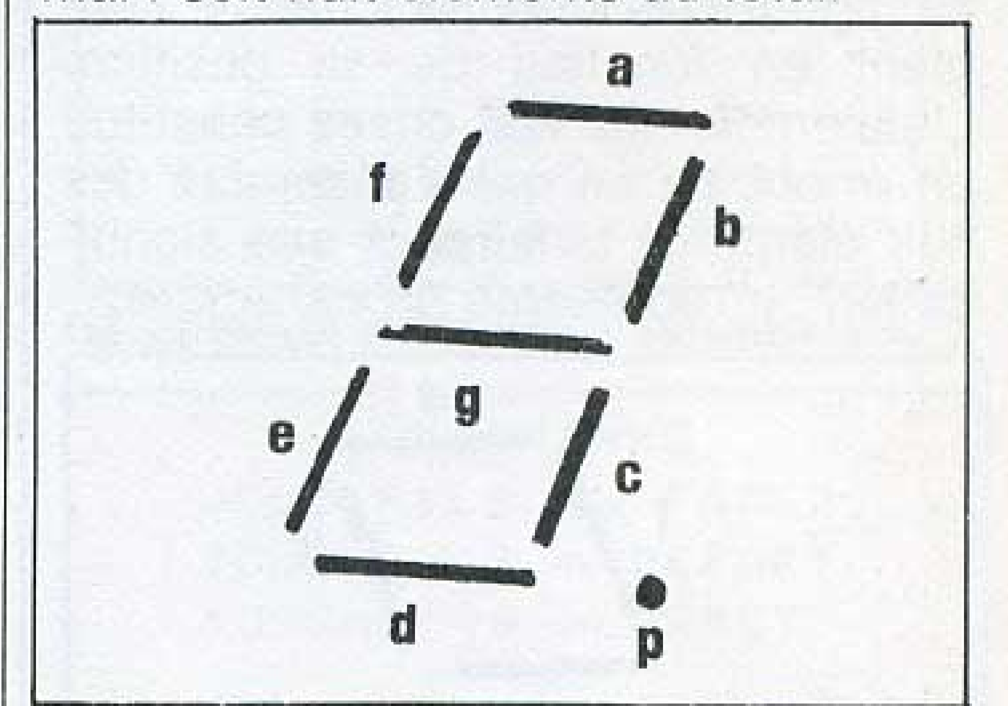


Fig. 15

Chaque segment ou barre, repéré par une lettre minuscule ; a, b, c, d, e, f, g et p pour le point décimal, est constitué d'1 diode électroluminescente. Une diode LED (Light Emitting Diode) a la particularité d'émettre une lumière dans le spectre visible quand elle est traversée par un courant de quelques dizaines de milliampères.

Le graphisme désiré, chiffre ou lettre, s'obtient en faisant passer un courant dans les segments qui doivent être allumés, tandis que les autres restent éteints.

Pour obtenir la lettre «H», il faut que les segments b, c, e, f et g soient allumés.
Comment allons-nous traduire cela dans le langage du micro-ordinateur ?
L'afficheur (fig 16) se compose de huit éléments binaires (7 segments et le point décimal), donc la représentation d'un graphisme à l'aide d'un octet paraît tout naturel. Selon la lettre ou le chiffre que l'on doit visualiser, il suffit d'allumer (état «1» logique) les segments correspondants. Les autres restent éteints (état «0» logique).
Ainsi la figure 16 donne une représentation binaire de la lettre H.

«L» est : 1000 0101 (segments allumés d, f et c) etc...
La représentation binaire, telle que nous venons de la voir, n'est pas très pratique pour l'utilisateur, même si elle est parfaitement compréhensible pour le micro-ordinateur. Aussi nous lui préférerons, très souvent, la représentation «hexadécimale».
Le code «HEXADECIMAL» consiste à remplacer chaque groupe de 4 éléments binaires (exemple 1000) par un caractère unique (8 pour 1000).
Comme 4 éléments binaires donnent naissance à 16 combinaisons différentes ($2^4 = 16$), nous obtenons le tableau de correspondance de la figure 18.

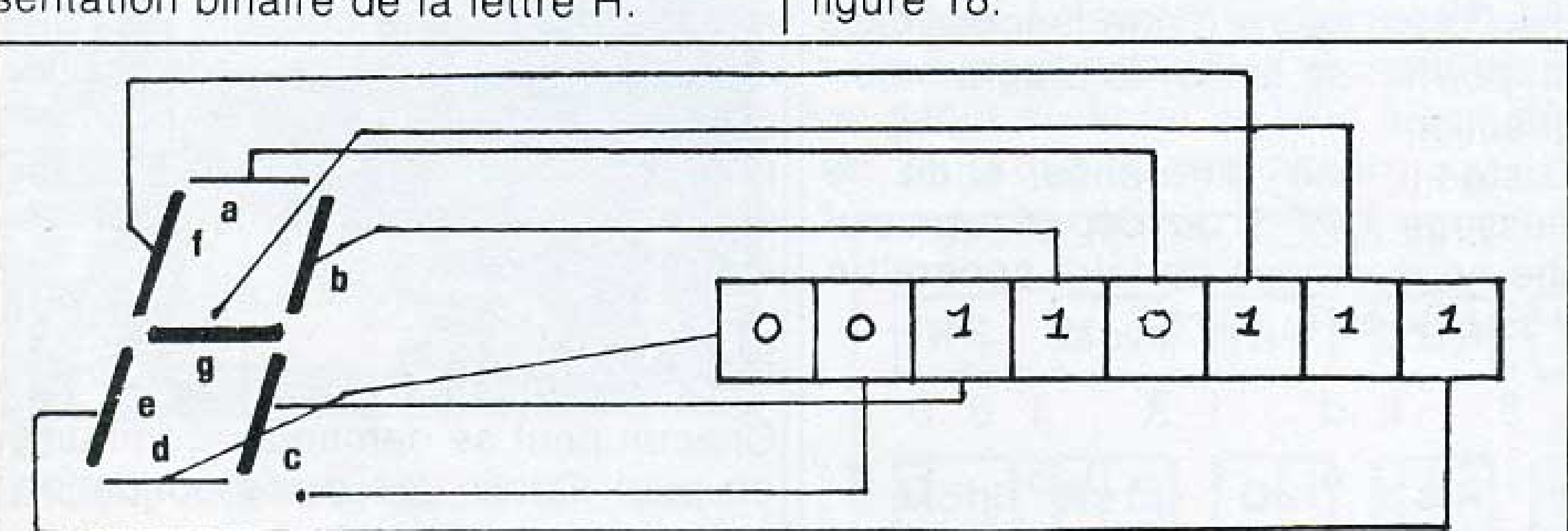


Fig. 16

Chacune des huit cases (fig 16 et 17) représente l'état d'un segment de l'afficheur ou point décimal. Lorsque le segment doit être allumé, on place un «1 logique», par contre s'il doit rester éteint on place un «0 logique».
La figure 17 identifie chaque segment en fonction de sa position. L'ensemble des huit cases constitue un «mot», parce que l'ensemble des huit éléments binaires a une signification : il représente un caractère.

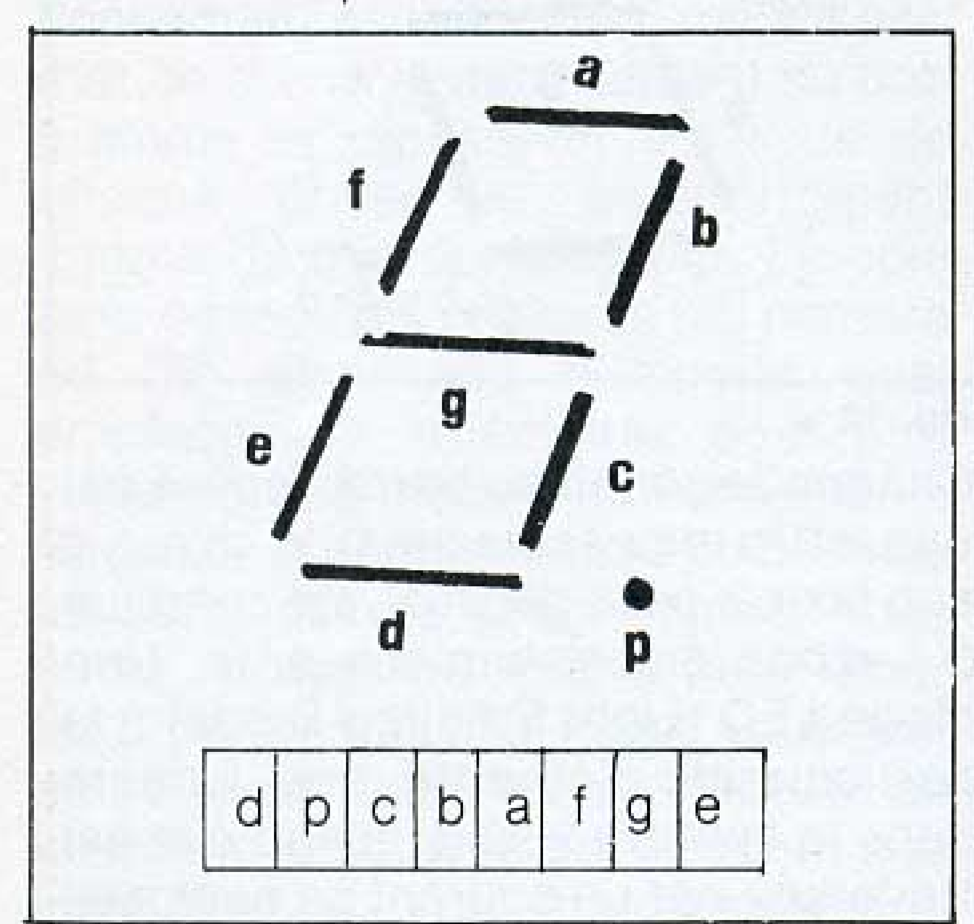


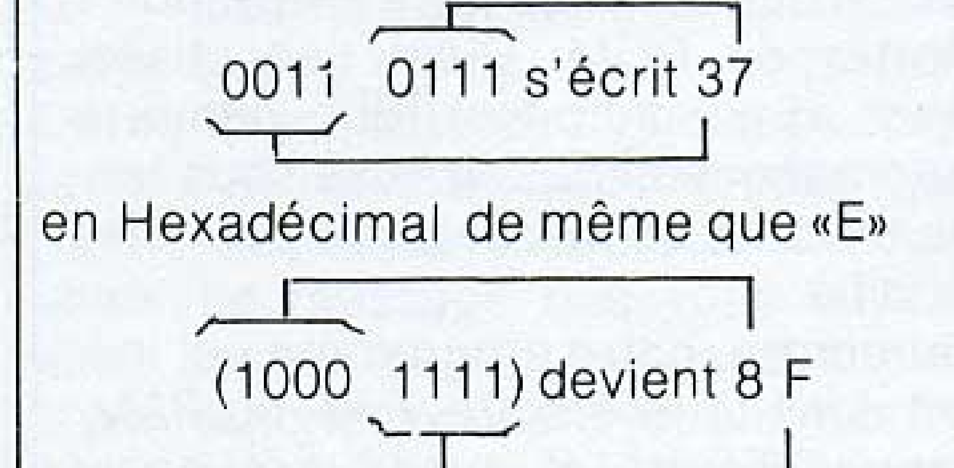
Fig. 17

Le code binaire de «H» est 00110111. (fig 16). De même, on en déduit que celui de :
«E» est : 1000 1111 (segments allumés d, a, f, g et e)

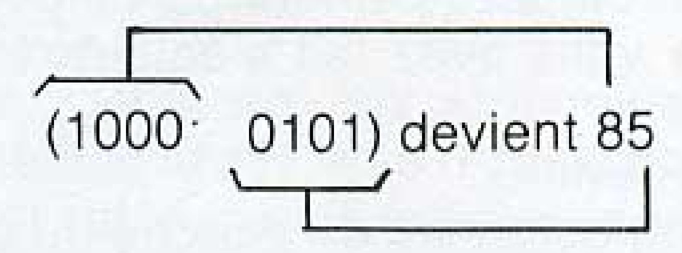
Equivalence binaire-hexadécimal			
Binaire	Hexa-décimal	Binaire	Hexa-décimal
0 0 0 0	0	1 0 0 0	8
0 0 0 1	1	1 0 0 1	9
0 0 1 0	2	1 0 1 0	A
0 0 1 1	3	1 0 1 1	B
0 1 0 0	4	1 1 0 0	C
0 1 0 1	5	1 1 0 1	D
0 1 1 0	6	1 1 1 0	E
0 1 1 1	7	1 1 1 1	F

Fig. 18

Maintenant, en tenant compte du tableau de la figure 18, nous pouvons écrire que
La lettre «H» dont le code binaire est



de même «L»



Le tableau de l'annexe 3 (page 94 du Manuel Technique du MPF-1) donne le code (2 caractères hexadécimaux) pour les chiffres et les lettres de l'alphabet. Il s'agit de caractères pseudo-alphabétiques, car avec 7 segments, certains caractères se représentent difficilement, ce sont généralement les lettres peu employées comme K, X, Z etc.

III.6 Clavier

Le clavier est représenté par la figure 14.
Chaque fois que vous appuyez sur l'une des seize touches blanches, ou «touches hexadécimales», c'est en réalité «4 éléments binaires» qui sont transmis au micro-ordinateur. Celui-ci effectue automatiquement la transposition telle qu'elle figure dans le tableau figure 18. Ce qui est vrai pour l'Ecriture, l'est aussi pour la lecture. Le système, au lieu de vous indiquer les «données» sous forme binaire (sauf cas particulier), les affichent sous la forme hexadécimale.
Ainsi Ecriture et Lecture sont parfaitement «homogènes», beaucoup plus aisée à manipuler, et le risque d'erreurs considérablement réduit.
Cependant, cette commodité ne doit pas vous faire perdre de vue que le microprocesseur ne travaille qu'en logique «Tout» ou «Rien».
Examinons à nouveau le programme 1. Les lignes b) et c) sont constituées systématiquement de groupe de deux caractères hexadécimaux, soit donc 8 bits. C'est la caractéristique fondamentale du Z80[R] utilisé : c'est un «micro» 8 bits.
Le format de base des messages, qu'ils soient des données ou des instructions est toujours de 8 bits ou 1 octet.
Un octet peut représenter 256 combinaisons différentes ($2^8 = 256$), et s'il s'agit de nombres entiers décimaux des quantités de 0 à 255.
Pour augmenter les quantités (si $N > 255$) ou les combinaisons, il faut utiliser 2,3 ou plus encore d'octets.
Par exemple, 2 octets (soit 4 codes hexadécimaux) peuvent représenter des nombres de 0 à 65 535. Et avec 3 octets, la quantité de nombres qui

peuvent être représentés est multiplié par 256. Ce qui devient de plus en plus confortable (avec 3 octets, on peut représenter des entiers de 0 à 16.777.215).

En résumé

— Le microprocesseur «travaille» avec des éléments binaires notés conventionnellement «0» ou «1». Les éléments sont rassemblés par groupes de 8 bits et constituent un «octet».
— Pour des raisons de commodité, les octets sont représentés avec 2 caractères hexadécimaux équivalents, aussi bien pour l'écriture que pour la lecture.
— Le clavier possède (partie de droite) 16 touches blanches, notées 0, 1,... 8, 9, A, B, C, D, E et F. Chacune d'entre elles correspond à un code hexadécimal, qui équivaut à quatre éléments binaires comme l'indique le tableau de la figure 18.
— Un octet (8 bits) est équivalent à 2 caractères (0 à F) hexadécimaux.
Exemples :

1011 1101 → BD
0011 1111 → 3F

IV. FAISONS LE POINT

Cette première partie a pour objectif de présenter l'architecture d'un système micro-ordinateur et comme nous voulons rester «concrets» pour être efficaces, nous nous sommes basés sur le MICROPROFESSOR MPF-1B qui nous accompagnera tout au long de notre étude.
Dès le prochain numéro, nous aborderons le «vif du sujet» : c'est-à-dire la conception, «hardware» proprement dite du Z80[R] et le déroulement des instructions à l'intérieur du microprocesseur.
Les quelques exercices proposés ont pour but de vous familiariser avec les différentes commandes du MPF-1 : introduire un programme, le modifier, l'exécuter, etc...
Si le contenu même des programmes, ce qui est probable, n'est pas toujours très explicite, c'est normal. Par contre, le rôle et la fonction de chaque mémoire ROM ou RAM doit être bien compris, après cette première partie.
Nous consacrerons plusieurs numéros pour le «langage du microprocesseur» et la méthode pour construire un programme. En attendant, voici quelques exercices... et un petit jeu de «PILE ou FACE» pour terminer.

V. EXERCICES

Exercice 1.
Nous utiliserons des cases mémoires qui se trouvent dans différentes zones (ROM, RAM, «vide») désignées par M. L'objet de cet exercice est de déterminer le type de mémoire en effectuant quelques opérations.
Séquence à effectuer :
a) Lire et noter le contenu des cases M à M + 4
b) Débrancher le MPF-1 (quelques secondes) et recommencer a)
Noter et comparer les résultats avec les précédents
c) Ecrire dans les cases M à M + 4, les valeurs successives 10, 11, 12, 13 et 14.
d) Relire le contenu des cases M à M + 4. Comparer avec ce que vous avez écrit au paragraphe c) ou éventuellement avec ce que vous avez lu au paragraphe a).
Différentes valeurs de M sont :
1. M = 0000
2. M = 1900
3. M = 0600
4. M = 1A00
5. M = 17FE
6. M = 2000
Indiquez les adresses qui appartiennent :
— à la mémoire ROM
— à la mémoire RAM
— à la zone «vide» (ni ROM ni RAM).

Exercice 2
Introduire le programme «Affichage et Clignotement» présenté dans le manuel technique du MPF-1, page 58.
a) Modifier le contenu de B adresse 180B (initiallement 32H) mettre 16H puis 64H. Que constatez-vous quand vous exécutez le programme ?
b) En ne modifiant que le contenu des cases 1826 à 182B faites clignoter que la lettre]-[. Puis la même chose pour les lettre «E» et «0» uniquement.
c) Modifier le contenu des cases mémoires 1820 à 182B pour faire apparaitre alternativement PILE et FACE sur les quatre afficheurs de gauche.

Exercice 3
JEU DE PILE OU FACE
Voici une version informatisée du «Pile ou Face».
Le programme introduit, il suffit de le lancer. Apparaissent alternativement sur les afficheurs les mots «PILE» ou «FACE», qui plus est se chevauchent à une cadence telle (30 changements par seconde) qu'il est impossible de les distinguer.
En appuyant sur la touche «0», le système se fige et indique soit «PILE» soit «FACE». Pour repartir appuyer sur la touche «3».
Les figures 19 et 20 indiquent respectivement l'Edition en langage «machine» et en langage «assembleur».

```
     PROGRAMME
JEU: PILE OU FACE?
TOUCHE   3 = DEPART
TOUCHE   0 = ARRET

1800 21 LD HL,1825
1803 E5 PUSH HL
1804 DD LD IX,1829
1808 DD EX (SP),IX
180A 06 LD B,03
180C CD CALL 0624
180F 30 JR NC,1815
1811 10 DJNZ 180C
1813 18 JR 1808
1815 FE CP 12
1817 20 JR NZ,1811
1819 CD CALL 05FE
181C FE CP 03
181E 20 JR NZ,1819
1820 18 JR 1808

1825 8F 8D 3F 0F
1829 00 00 8F 85
182D 30 1F
```

Fig. 20 Edition en langage « assembleur ».

```
     PROGRAMME
JEU: PILE OU FACE?
TOUCHE   3 = DEPART
TOUCHE   0 = ARRET

1800 21 25 18 E5
1804 DD 21 29 18
1808 DD E3 06 03
180C CD 24 06 30
1810 04 10 F9 18
1814 F3 FE 12 20
1818 F8 CD FE 05
181C FE 03 20 F9
1820 18 E6 76 4E
1824 41 8F 8D 3F
1828 0F 00 00 8F
182C 85 30 1F
```

Fig. 19 Edition en langage « machine ».

Philippe Duquesne

COURS PRATIQUE DE MICROPROCESSEUR AVEC LE MICROPROFESSOR MPF-IB

DEUXIEME PARTIE

Le hardware du Z-80

SOMMAIRE

I. INTRODUCTION

Dans la première partie, intitulée «Concept de Base» nous avons montré le rôle primordial du microprocesseur dans une calculatrice puis, d'une manière plus générale, dans un micro-ordinateur. En nous basant sur le microprofessor MPF-IB, nous avons présenté l'architecture du système complet, qui constitue le support pédagogique de notre cours.
Nous abordons, dans cette deuxième partie, la structure interne du Z80R, souvent désigné par «Hardware» par opposition au «Software» qui est l'étude du logiciel.
Cette connaissance «hard» n'est pas indispensable pour la programmation, mais elle y contribue beaucoup et permet de mieux comprendre les différentes étapes dans le déroulement des instructions.
C'est à l'aide de quelques exemples courts mais très instructifs, exécutés avec le MPF-IB, que vous découvrirez «pas à pas» la fonction des différents éléments essentiels du microprocesseur Z80R. Une fois encore, vous constaterez que théorie et pratique vont de paire en micro-informatique.
Nous terminerons la partie Hardware avec la présentation globale de l'organisation interne du Z80R sur laquelle chaque fonction essentielle est représentée par un «bloc fonctionnel».

Dans la première partie, nous avons vu que les deux «pièces maitresses» du C.P.U. sont d'une part l'Unité Arithmétique et Logique (U.A.L.) et d'autre part l'Unité de Contrôle (U.C.). A l'intérieur même du C.P.U. est étroitement associée à ces deux unités fondamentales, une zone de mémoire privilégiée, constituée d'un ensemble de 22 registres.
C'est à l'étude de cette mémoire et des instructions qui s'y raportent que nous consacrerons essentiellement cette deuxième partie.
A partir de cette nouvelle connaissance, nous comprendrons mieux les opérations qui se réalisent avec l'U.A.L.

II. LES REGISTRES

II.1. Rappel

Un registre est un circuit constitué d'un certain nombre de cellules «mémoire élémentaire», capables de mémoriser une information binaire. Comme le «format» des mots du Z80R est soit d'un octet ou deux octets, les registres sont constitués de 8 ou 16 cellules juxtaposées (voir Led Micro n°8 pages 62 à 71 pour plus de détail).
Les informations binaires sont inscrites dans le registre au cours d'une opération de «chargement». Toute nouvelle écriture écrase la précé-

dente. Par contre, l'opération de lecture n'altère pas le contenu de la mémoire. Le fonctionnement des registres du C.P.U. s'apparente à celui des mémoires RAM. Ce qui est normal, car en fait leur technologie de réalisation est similaire. Le contenu des registres disparait en cas de coupure d'alimentation, et le contenu de ceux-ci est aléatoire à la mise sous tension.

L'ensemble des registres du Z80[R], qui constitue l'une des originalités de ce composant peut être scindé en deux groupes :

— les registres d'usage généraux
— les registres «pointeurs d'adresses».

L'ensemble des registres généraux sont des 8 bits (comme le «format» du microprocesseur) et ils interviennent principalement dans la «manipulation» de données. Ils jouent le rôle de «mémoire tampon» à accès rapide pour fournir un opérande, ou mémorisent les résultats intermédiaires des opérations. Tout en étant des registres 8 bits, ils peuvent, sous certaines conditions, être associés par paire et ainsi constituer un registre de 16 bits «unique».

Le second groupe ne comporte que des registres 16 bits dont le rôle essentiel est celui de «pointeur d'adresses» dans la zone mémoire. Au moins l'un d'eux intervient chaque fois qu'un accès à la mémoire est nécessaire. Compte tenu du rôle de ce second groupe, on conçoit aisément que tous ces registres soient connectés directement sur le bus d'adresses, et que leur taille soit de 16 bits (comme celle du bus d'adresses).

Nous examinerons successivement les différents registres individuellement ou groupés par fonctions. Cependant, l'un d'entre eux joue un rôle primordial, c'est l'ACCUMULATEUR.

C'est par ce dernier que nous commençons notre étude.

II.2. L'Accumulateur

L'Accumulateur ou registre A est étroitement lié à l'Unité Arithmétique et Logique. L'U.A.L. renferme les différents circuits logiques nécessaires à la réalisation des opérations arithmétiques et logiques. L'instruction spécifie la fonction qui doit être accomplie. Les quantités ou opérandes sur lesquelles porte l'opération à

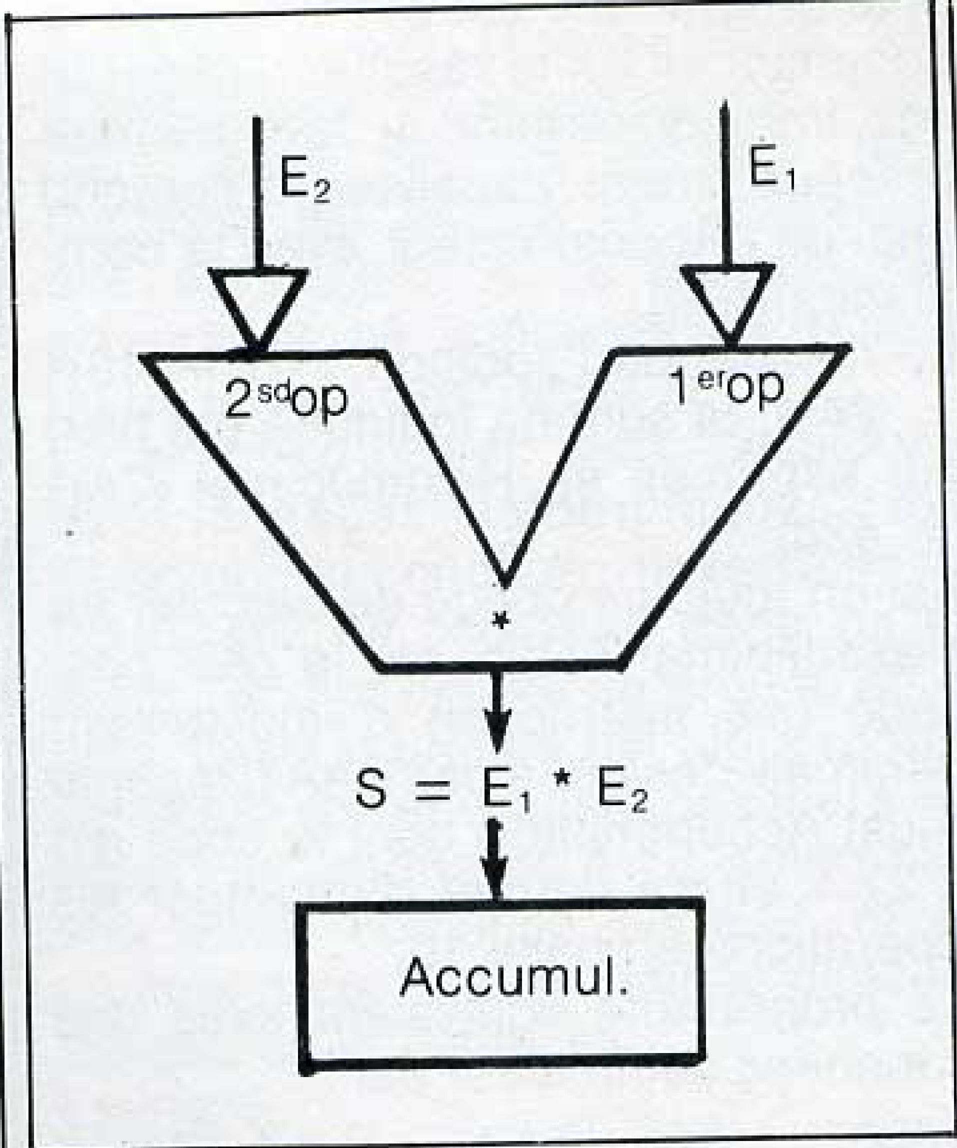


Fig. 21

effectuer sont placées sur chacune des entrées E1 et E2 de l'U.A.L. (fig. 21). Le résultat apparait en sortie. En désignant par *, la fonction (* désigne l'addition, la soustraction le ET logique, etc...) le résultat est tel que S = E1 * E2.

Comme l'U.A.L. est un ensemble combinatoire, il ne peut conserver le résultat S, il faut lui adjoindre un élément mémoire qui est le registre A.

L'Accumulateur est un registre 8 bits dans lequel l'U.A.L. dépose le résultat des opérations arithmétiques et logiques. Ainsi l'Accumulteur apparait comme le registre «destinataire» des opérations, tout au moins celles qui utilisent des données sous la forme d'un octet.

Bien que cela puisse paraître curieux de prime abord, le contenu du registre A constitue pour de nombreuses opérations sur 8 bits l'un des opérandes.

Dans ce cas, le registre A joue à la fois la «source» d'une des données et la «destination» du résultat. Illustrons ce double rôle, en réalisant une addition de deux nombres.

Supposons que nous voulions réaliser l'opération suivante : additionner 17 et 25 ; placer le résultat dans A.

Nous présentons tout d'abord le cheminement des opérations (fig 22 a) et b)) tel qu'il s'effectue dans le C.P.U. et nous indiquons le programme correspondant.

La première opération consiste à charger l'Accumulateur avec la quantité 17 (décimal) (fig 22 a)). Le microprocesseur n'utilise que des valeurs hexadécimales : donc il faut

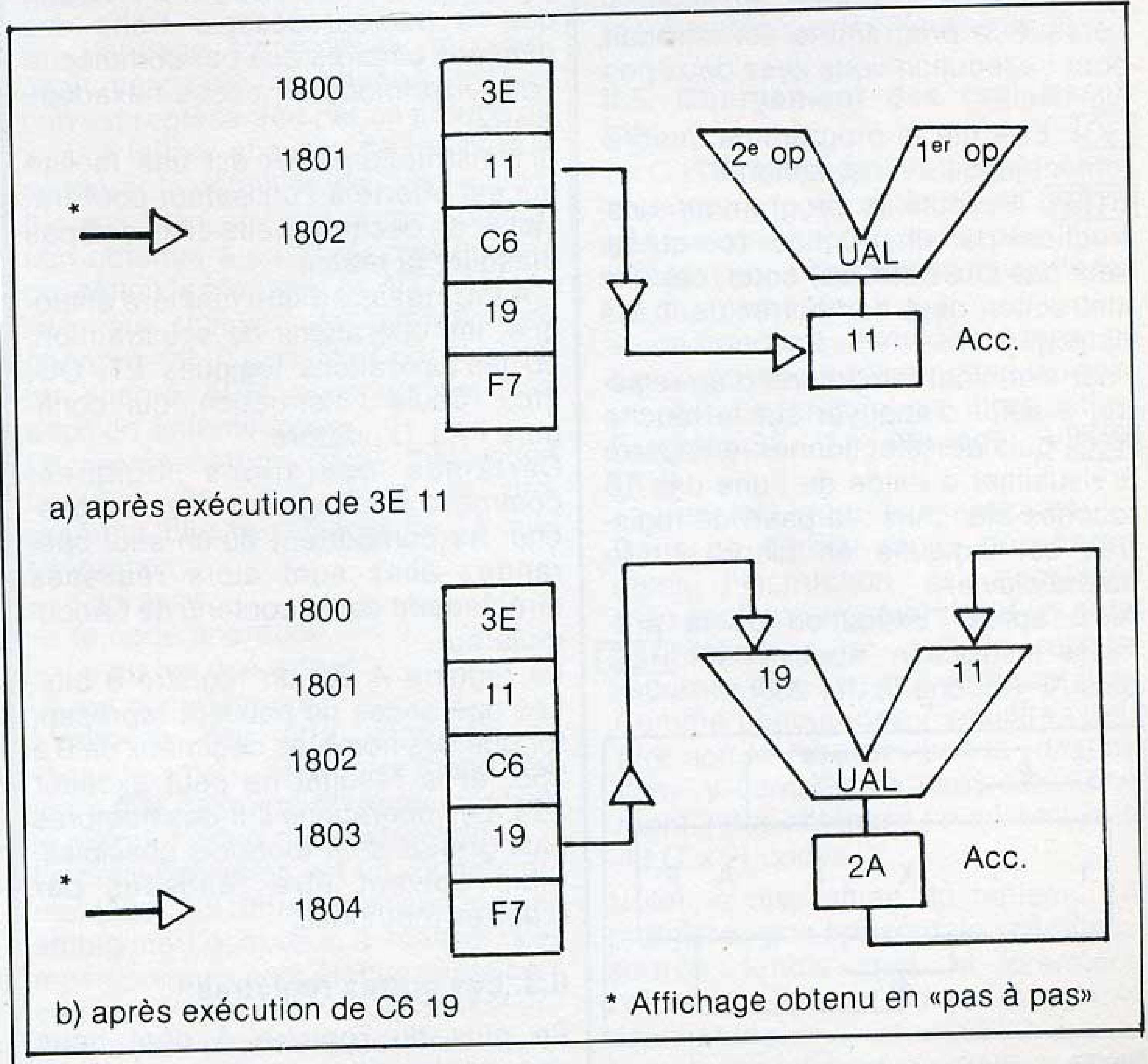


Fig. 22

convertir 17 d en «hexa» ce qui donne 11H (voir le tableau d'équivalence dans le Manuel Technique du MPF-I, page 99). L'instruction qui permet le chargement du registre A est 3E. Elle doit être suivie immédiatement de la «donnée», c'est-à-dire l'octet à placer dans le registre A.

Nous obtenons ainsi 3E 11, pour la première instruction. La seconde donnée, peut provenir d'un registre, d'une case mémoire extérieure ou être contenue dans l'instruction ADD elle-même.

L'instruction ADD (ADDition) dont le code est C6 additionne au contenu de l'Accumulateur (1er opérande) l'octet qui suit immédiatement le code opératoire. Le résultat est déposé dans l'Accumulateur (fig 22 b). Comme pour le premier opérande, la quantité 25 que nous voulons additionner à 17 doit être convertie en «hexa» ce qui donne 19H. Nous obtenons ainsi le programme complet, que vous introduisez à partir de 1800 :

Adresse	Code	Hexa	Instruction
1800	3E	11	Ld A.17d
1802	C6	19	ADD A.25d
1804	F7		Fin

Lorsque le programme est introduit, pour l'exécution vous avez deux possibilités :

GO: Exécute le programme intégralement jusqu'à l'instruction F7.

STEP: Exécute le programme «instruction» par «instruction» (ce qui ne veut pas dire octet par octet, car une instruction peut comporter de 1 à 4 octets).

Pour examiner le contenu d'un registre, il suffit d'appuyer sur la touche REG puis de sélectionner le registre à visualiser à l'aide de l'une des 16 touches blanches : la paire de registres est indiquée en blanc sur le cache-clavier.

Ainsi après l'exécution de la première instruction, appuyer sur REG puis AF (touche 0), fig 23.

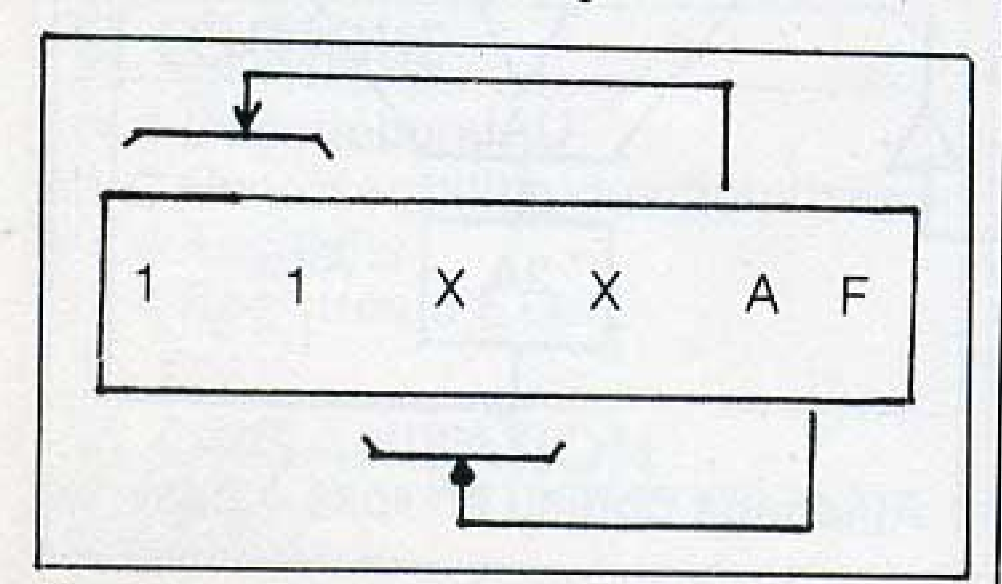


Fig. 23

Si le programme est exécuté avec la commande GO le résultat n'apparait pas immédiatement. Il faut comme précédemment visualiser le contenu final de l'Accumulateur avec la commande REG

Le résultat de l'addition 11H + 19H = 2AH, la somme indiquée est bien sûr exprimée en Hexadécimal (2AH = 42d).

Est-on toujours obligé de calculer en Hexadécimal ? Non, car le Z80R possède une instruction d'«ajustement décimal» désignée par DAA (Decimal Ajust Accummultor) dont le code est « 27 » et qui permet d'effectuer les opérations en base 10.

Le programme équivalent avec des «données décimales» est :

1800	3E	17	Ld A.17d
1802	C6	25	ADD A.25d
1804	27		DAA
1805	F7		FIN

Après l'exécution de l'instruction 1802, le contenu de A est 3C il correspond effectivement à l'addition de 17H et 25H. La quantité 42 (17 + 25 = 42) n'apparaît dans A qu'après l'exécution de l'instruction d'ajustement (DAA).

Le lecteur ne doit pas perdre de vue que le microprocesseur traite des données binaires que par commodité nous exprimons en codes hexadécimaux.

Si l'instruction DAA est une facilité qui est offerte à l'utilisateur pour travailler en décimal, celle-ci ne doit pas masquer la réalité.

L'A.L.U. réalise d'une manière analogue, les opérations de soustraction, ou les opérations logiques ET, OU, etc... Seule l'instruction, qui configure l'A.L.U., diffère.

Certaines opérations logiques, comme le décalage à droite ou à gauche, ne comportent qu'un seul opérande : elles sont alors réalisées directement sur le contenu de l'Accumulateur.

Le registre A est un registre 8 bits. Les opérandes ne peuvent représenter que des nombres décimaux de 0 à 255, et le résultat ne peut excéder 255. Les opérations sur des nombres plus grands sont toutefois possibles, mais doivent être réalisées par d'autres méthodes.

II.3. Les autres registres

En plus du registre A dont nous venons de définir le rôle, six autres registres, dits «registres généraux» sont accessibles dans le C.P.U.

La figure 24 donne la représentation couramment employée. Ces six registres sont respectivement désignés par les lettres B, C, D, E, H et L sans que celles-ci n'aient d'autre signification que de suivre l'ordre alphabétique. Tout comme le registre A, chacun d'eux peut contenir un octet, soit une quantité décimale de 0 à 255, soit une quantité hexadécimale de 00 à FF.

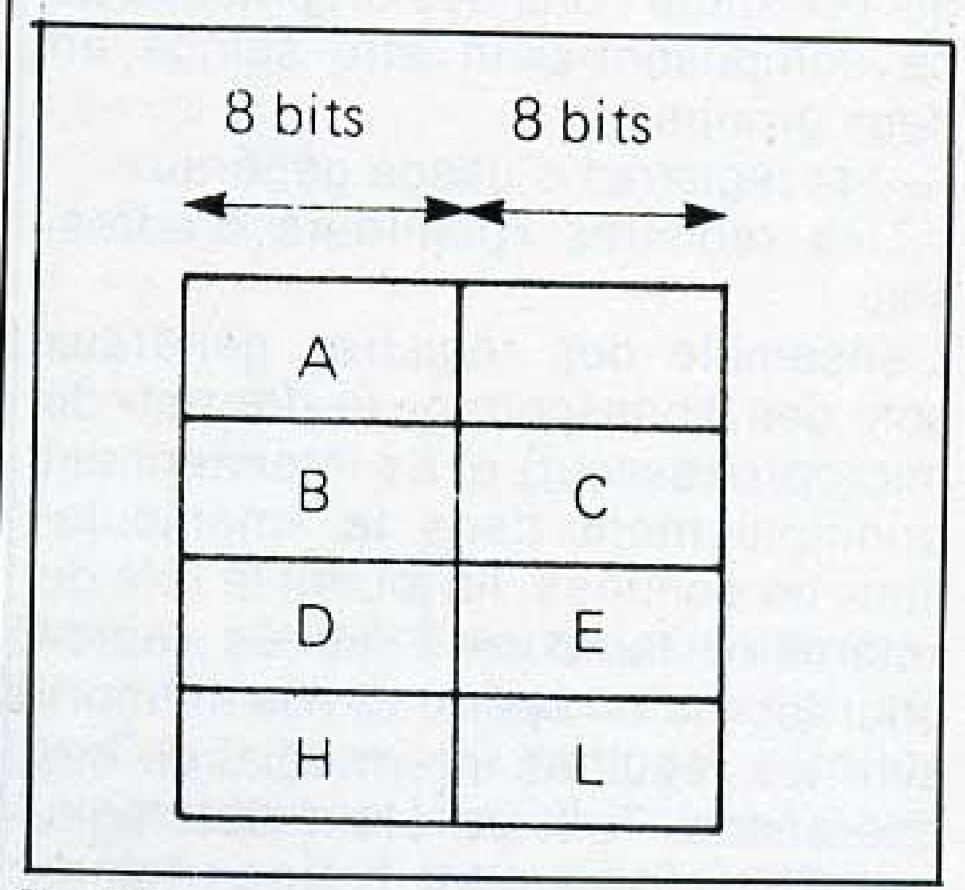


Fig. 24

Les emplois de ces registres sont nombreux et nous le découvrirons petit à petit au fur et à mesure de la progression du cours.

Nous pouvons cependant dire d'ores et déjà qu'ils sont utilisés pour conserver des résultats intermédiaires ou pour des stockages momentanés.

Leur emploi est souvent préféré à un emplacement mémoire (dans la RAM extérieure) car leur «temps d'accès» est plus rapide et la mise en œuvre plus simple et plus maniable. Par exemple, le temps nécessaire au stockage du contenu de l'Accumulateur (1 octet) dans une case mémoire extérieure est trois fois supérieur que pour stocker dans l'un des registres auxiliaires. Inutile de dire, que la réciproque (restitution d'un octet dans l'Accumulateur) est aussi trois fois plus rapide avec l'emploi des registres généraux.

En examinant la figure 24, le lecteur peut penser, à priori, que la disposition ainsi adoptée est due à la «fantaisie» de l'auteur et qu'une autre disposition, plus conforme à l'ordre alphabétique par exemple, aurait été aussi valable.

La suite justifie et explique cette configuration.

Il arrive fréquemment que l'information à traiter occupe plus d'1 octet. Nous avons déjà rencontré, par exemple, ce problème dans le cas de l'adressage de la mémoire, où nous avons utilisé un format 2 octets (ou 16 bits). Pour pouvoir disposer ainsi de registres doubles, on associe deux registres 8 bits pour constituer une paire de registres, qui se comporte dès lors comme un unique registre 16 bits (ou 2 octets). Ainsi les registres B et C peuvent être associés pour former la paire BC, D et E pour former la paire DE et enfin H et L pour former HL.
La figure 25 donne une nouvelle représentation des registres généraux associés par paires.
Les opérations sur 16 bits deviennent aussi aisées que pour 8 bits. Le «nom» et surtout le code de l'instruction changent.

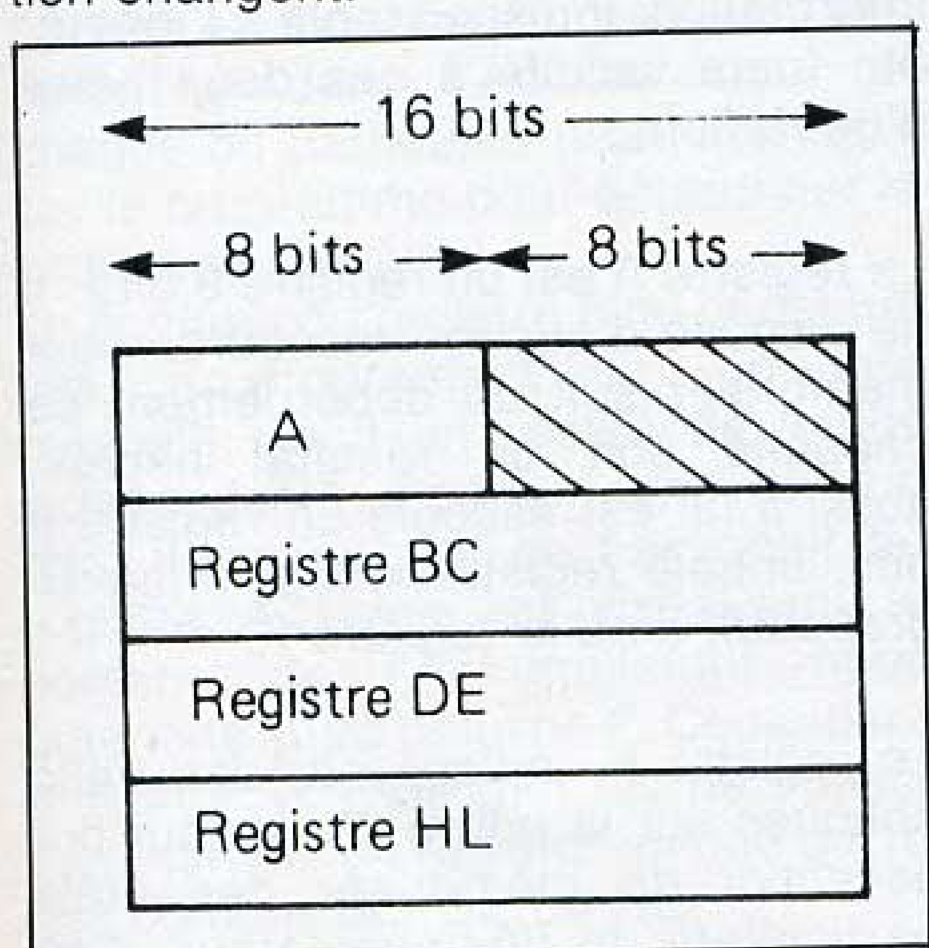


Fig. 25

Dans cette nouvelle configuration des registres, l'A.L.U. réalise des opérations arithmétiques et logiques non plus sur 8 bits mais 16 bits en une seule instruction. Etant donné qu'il n'est pas prévu d'associer au registre A, le registre adjascent bien qu'existant (nous verrons bientôt son rôle), **la paire de registres HL joue, dans les opérations double octets, le même rôle que l'Accumulateur dans les opérations sur 8 bits.** Par exemple le contenu de la paire de registres HL **peut être additionné avec le contenu** de la paire de registres DE. Le résultat est déposé dans la paire de registres HL.

II.4. Représentation de l'instruction

Le seul langage qui puisse être compris par un microprocesseur est le «langage binaire». Introduire tout un programme en binaire (avec des interrupteurs par exemple) est faisable ; outre le fait qu'une telle méthode est longue et fastidieuse, elle est sujette à de nombreuses erreurs et ne présente en réalité qu'un intérêt très restreint.
Le premier pallier dans l'évolution des langages informatiques est l'emploi du codage hexadécimal. Pour cela, on scinde, à partir de la droite, le mot binaire en groupe de 4 bits ; chaque groupe est alors représenté par son équivalent décimal si la valeur est inférieure à 10, sinon par une lettre comprise entre A et F comme nous l'avons indiqué dans le tableau de la figure 18.
Exemple :
0011 1110 ⟶ 3E
1100 0110 ⟶ C6
C'est ce que nous avons fait, et les différents programmes que nous avons réalisés étaient exprimés en «hexadécimal», encore appelé «langage machine».
Cependant, lors de l'élaboration d'un programme, les informations peuvent être codées sous une forme plus pratique et surtout plus compréhensible par l'utilisateur, le «codage symbolique».

En «langage symbolique» ou «langage assembleur», chaque instruction est représentée par un groupe de 3 ou 4 lettres choisies de manière à suggérer la définition de l'instruction. Il s'agit bien souvent d'une abréviation obtenue à partir du mot anglais qui définit la fonction. Ainsi LD est utilisé pour signifier une opération de chargement (Load = Chargement) ou encore ADD pour réaliser une addition arithmétique.
La représentation d'une instruction occupe une ligne dans un programme. Elle se compose de 2 à quatre parties :
— l'adresse
— le code opération
— le ou les opérandes
— les commentaires
L'adresse doit être exprimée en «hexadécimal». Elle correspond toujours à la case mémoire qui contient le premier octet de l'instruction.
Le code opération est un «code mnémonique» qui détermine sans aucune ambiguité l'opération à réaliser. Les mnémoniques sont établis par le concepteur même du microprocesseur.
Le «code mnémonique» est suivi, éventuellement, d'opérandes qui spécifient et décrivent les données sur lesquelles porte l'instruction. Les indications fournies peuvent être telles que la source et/ou la destination des données ou du résultat. Selon le type de l'instruction, celle-ci nécessite un ou plusieurs opérandes. Quelquefois aucun : DAA par exemple.
Les instructions peuvent être accompagnées de commentaires ; ce sont des renseignements complémentaires relatifs au programme qui ne servent qu'à l'utilisateur en rappelant le but de l'étape.
Le programme symbolique ainsi obtenu, bien que très évocateur, ne peut être compris tel quel par le microprocesseur, qui ne traite que des informations binaires. Il apparait donc indispensable de traduire le programme symbolique en un programme binaire, ou tout au moins exprimé en hexadécimal, c'est le code machine. Pour des raisons de commodité de représentation, l'expression des instructions en hexadécimal est inscrite entre l'adresse et le mnémonique.
Pour faciliter la phase de traduction, des mnémoniques en code hexa, nous utiliserons des tableaux par groupes de fonctions comme ceux représentés par les figures 26 et 27.

II.5. Chargement des registres 8 bits

Le C.P.U. possède 7 registres (Accumulateur + 6 registres d'usages généraux) chacun de 8 bits. Pour l'instant nous étudierons deux modes de chargement :
— chargement immédiat, dans ce cas la donnée à placer dans le registre est l'opérande de l'instruction. Exemple 3E 11 (changer A avec 11H)
— transfert entre deux registres
Dans ce dernier mode de chargement, l'instruction est constituée d'un seul code opératoire d'un octet qui définit d'une part le registre «source» et le registre «destinataire». Comme chacun des 7 registres peut être soit la «source» soit la «destination», y compris les deux simultanément, nous obtenons un ensemble de 49 (7 x 7) codes.
Noter la disposition du tableau. La première ligne horizontale spécifie la source tandis que la première colonne de gauche indique le registre destinataire.
La colonne à l'extrême droite indique

		SOURCE						
	A	B	C	D	E	H	L	IMMED. n
A	7F	78	79	7A	7B	7C	7D	3E n
B	47	40	41	42	43	44	45	06 n
C	4F	48	49	4A	4B	4C	4D	0E n
D	57	50	51	52	53	54	55	16 n
E	5F	58	59	5A	5B	5C	5D	1E n
H	67	60	61	62	63	64	65	26 n
L	6F	68	69	6A	6B	6C	6D	2E n

DESTINATION

Fig. 26 : Instruction « LD ».

les codes correspondants au chargement immédiat des registres. Dans ce cas, le code opératoire est suivi de l'octet à charger désigné par «n». Sachant que «n» représente une quantité décimale comprise entre 0 et 225 ou 00 et FF en hexadécimal.
Application. Charger le registre B avec 37H. Recopier cette donnée dans le registre C.
Le programme est :

1800	06	37	Ld B.37
1802	48		Ld C,B
1803	F7		FIN

A noter que dans la seconde instruction, la destination précède la source.
Introduisez le programme sur votre MPF-1 et vérifiez le contenu des registres B et C.

II.6. Opérations Arithmétiques simples

L'instruction ADD A, n réalise l'addition de la quantité hexadécimale n au contenu de A et place le résultat de l'opération dans l'Accumulateur.
L'instruction SUB A, n réalise la soustraction. La quantité n est soustraite du contenu de A et le résultat est replacé dans l'Accumulateur.
Les deux opérations arithmétiques addition et soustraction peuvent être réalisées, non plus avec une quantité «n» déterminée par l'octet qui suit le code opératoire mais avec le contenu de l'un quelconque des registres (y compris le registre A lui-même). Le premier opérande est dans tous les cas le contenu de A, et le résultat de l'opération est toujours déposé dans l'Accumulateur.
Le tableau de la figure 27 résume les instructions d'addition et de soustraction en indiquant le code machine correspondant.

Si la représentation utilisée n'est pas tout à fait correcte du point de vue syntaxe (de l'assembleur), elle présente l'avantage d'être très évocatrice.
Ainsi A ← A + C, qui se traduit en langage assembleur par ADD A, C, signifie : ajouter au contenu du registre A celui de C et déposer le résultat dans le registre A.
Comme dans le cas de l'addition, la soustraction est réalisée par le CPU sur des quantités exprimées en héxadécimal. Toutefois, l'instruction d'ajustement décimal DAA (code 27) joue un rôle indentique pour permettre des soustractions sur des quantités décimales.

Exemple

1800	3E	37	Ld A,37
1802	06	12	Ld B,12
1804	90		SUB A, B
1805	27		DAA
1806	F7		FIN

Après exécution de ce programme, le contenu de A est 25.

II.7. Registre F.

Dans le paragraphe précédent, nous avons montré comment réaliser des opérations arithmétiques. Mais à aucun moment nous n'avons pris en considération, le fait que le résultat d'une addition peut être supérieur à la capacité de l'Accumulateur (>FFH ou 255d), ni non plus que la soustraction donne un résultat négatif si le second opérande est supérieur au premier. Or il s'agit d'une information indispensable : l'ignorer ote toute validité à ces deux types d'opérations.

Le registre A est un registre 8 bits ; il ne dispose d'aucune ressource pour mémoriser soit un débordement de capacité soit un résultat négatif, aussi il lui est associé un registre 8 bits, appelé registre «Flag» (Flag = drapeau), d'où le registre F.

Le registre F, indiqué par des traits obliques sur la figure 25, a pour but essentiel de mémoriser des états particuliers de l'Accumulateur, d'où le nom qui lui est aussi donné Registre d'ETATS.
La configuration du registre est indiquée par la figure 28.
Par exemple, la case 7 contient la lettre «S» pour signe. Ce bit passe à 1 quand le résultat d'une opération est négatif. Dans le cas contraire, il est à «0». La case «6» contient la lettre «Z» pour zéro. Cet indicateur est positionné en 1 quand une opération arithmétique ou logique entraine un résultat nul (00) dans l'Accumulateur. Dans le cas contraire (résultat différent de 0), il est remis (0 logique).

Registres	A	B	C	D	E	H	L
Addition	A←A+A	A←A+B	A←A+C	A←A+D	A←A+E	A←A+H	A←A+L
Code	87	80	81	82	83	84	85
Soustracteur	A←A−A	A←A−B	A←A−C	A←A−D	A←A−E	A←A−H	A←A−L
Code	97	90	91	92	93	94	95

Fig. 27 : Instructions ADD et SUB.

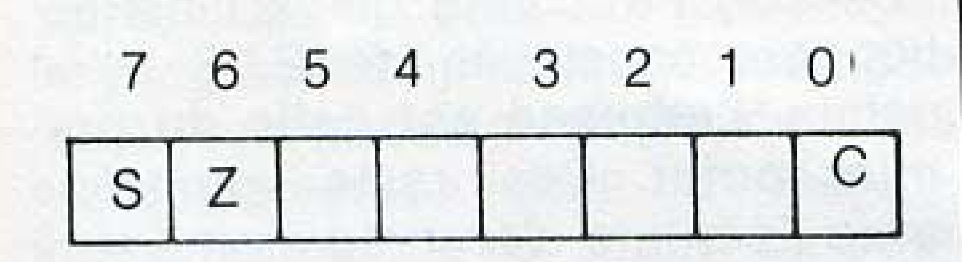


Fig. 28

La case «0» contenant la lettre «C» pour Carry (ou dépassement) passe à 1, si le résultat d'une opération dans l'Accumulateur entraine un dépassement de capacité. Ainsi l'addition de 187 et de 78 (187 + 78 = 265) donne un résultat supérieur à 255. L'Accumulateur contiendra 10 (265 − 255 = 10) et le bit 0 du registre F sera en 1.

Ainsi le registre F contient «8 drapeaux» (Flags) qui sont levés (positionnés ou en 1) ou baissés (remis au 0). En réalité six seulement sont utilisés (bit 3 et bit 5 ne sont pas utilisés). Les flags sont automatiquement positionnés en fonction du résultat d'une opération dans l'Accumulateur et chaque bit particulier peut être testé par le programme pour déterminer la séquence à suivre.

Le contenu du registre F ne constitue pas, à proprement dit, un mot puisqu'il s'agit en réalité de 8 bits indépendants. Compte tenu de son rôle, il est normal de l'associer au registre A ; ainsi en visualisant le contenu de l'Accumulateur nous obtenons aussi celui de F. Cependant si A et F vont de pair, ils ne constituent jamais un registre unique de 16 bits comme BC, DE ou HL.

Etant donné la spécificité du registre F, le positionnement des flags peut apparaitre directement sur l'affichage. La commande REG suivie de «C» visualise les états des 4 bits de gauche (S, Z, X, H) fig 27 a) tandis que REG suivie de «D» indique les états des 4 bits de droite (X, P/V, N, C) fig 27 b).

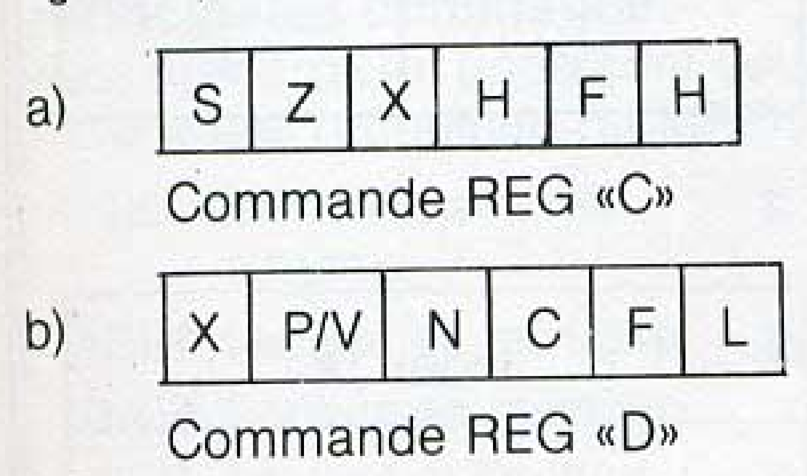


Fig. 29 : Visualisation d'un registre d'ETATS.

Nous reprendrons plus en détail, la signification de chaque bit du registre d'Etats quand nous décrirons l'Unité Arithmétique et Logique

II.8. Second jeu de registres

Nous avons passé en revue les registres généraux du C.P.U. Nous avons appris qu'il existait un registre Accumulateur A et six registres généraux B, C, D, E, H et L et un registre d'états F. Bien que cet ensemble soit suffisant dans la plupart des opérations, il existe, en réalité, un deuxième ensemble, rigoureusement identique, image du premier et désignés par A', B', C', D', E', H', L' et F', comme l'indique la figure 30.

Toutefois, à un instant donné, un groupe et un seul groupe de 8 registres (Prime ou non Prime) parmi les deux peut être utilisé. C'est-à-dire que le jeu de registres utilisé peut être «Lu» ou «Ecrit», tandis que l'autre joue le rôle de mémoire uniquement.

Le choix s'effectue au moyen de deux instructions distinctes. L'une sélectionne l'ensemble A et F ou A' et F' tandis que l'autre sélectionne les registres B' à L ou B' à L'..

Quel est l'avantage d'avoir doublé l'ensemble des registres accumulateurs et généraux ? Nous le comprendrons mieux lorsque nous étudierons les «interruptions».

Cependant nous pouvons dire dès à présent qu'il arrive qu'au cours de l'exécution d'un programme, le microprocesseur soit amené à traiter une autre tâche plus prioritaire que celle en cours. Il suspend alors le programme principal, le laissant tel quel dans les registres A, B,... H et L et traite la «tâche urgente» à l'aide des registres A', B',... H' et L'. Cette opération étant accomplie, le C.P.U. reprend le déroulement du programme principal, là où il l'avait interrompu.

Les «interruptions» sont très fréquemment utilisées pour répondre à des demandes extérieures mais elles restent toujours sous le contrôle du programme, donc de son auteur. C'est d'ailleurs un des points forts du Z80, son aptitude à gérer les interruptions.

Enfin nous trouvons deux autres registres 8 bits qui ne servent que dans des cas bien spécifiques.

Le registre I est utilisé dans certains modes d'interruption.

Le registre R, exclusivement sous le contrôle du C.P.U. est utilisé pour le rafraîchissement (R = Refresh) automatique des mémoires du type RAM

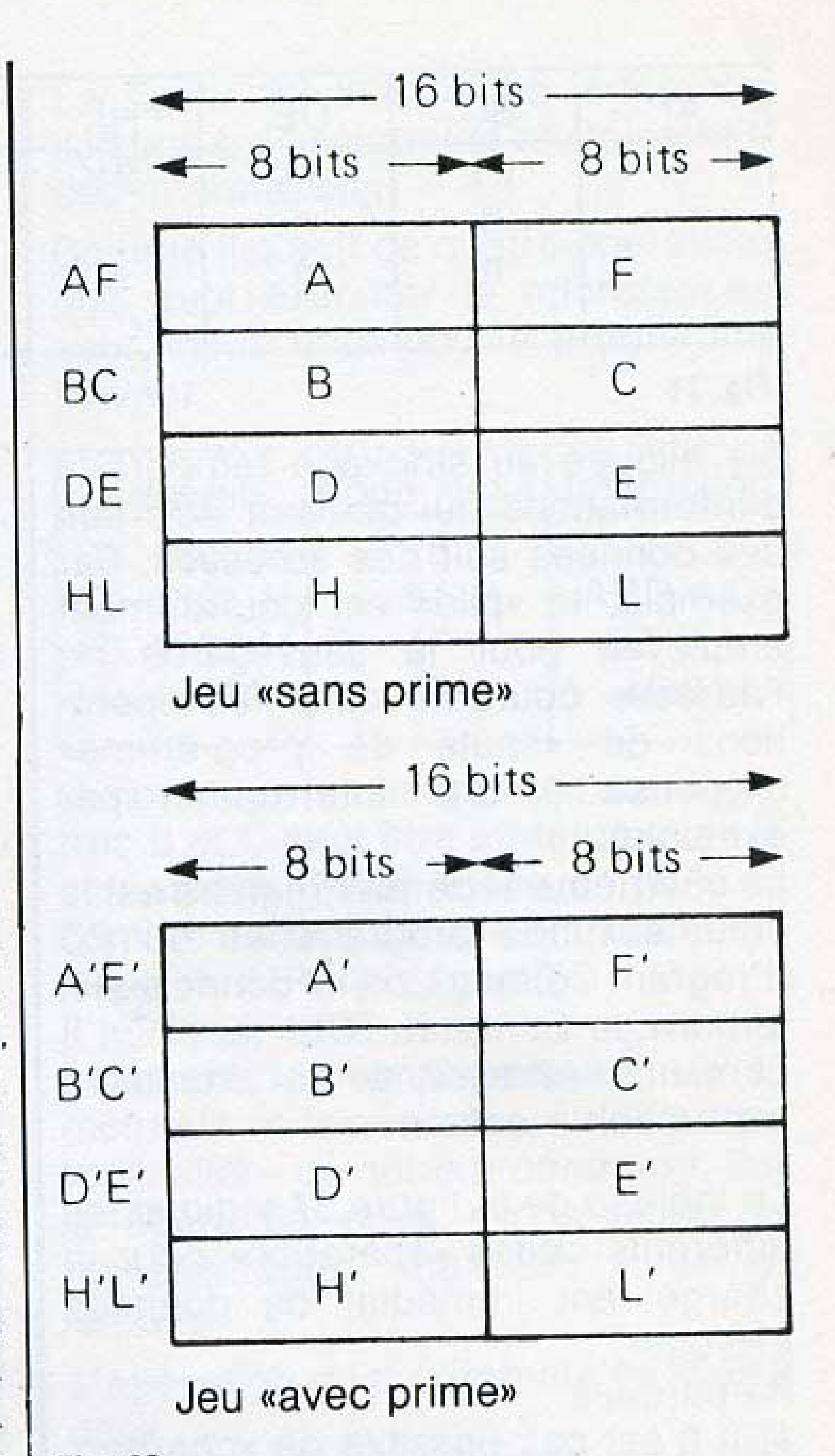


Fig. 30

dynamiques. Les RAM's statiques ne doivent pas être rafraîchies.

II.9. Registres 16 bits

Le C.P.U. possède encore quatre autres registres, exclusivement de taille 16 bits, essentiellement destinés à l'adressage de la mémoire ou des périphériques connectés sur le bus d'adresse.

Deux registres respectivement appelés Registre IX et Registre IY sont des registres d'Index. Cette technique d'adressage, confère au système une grande souplesse, notamment quand il faut accéder aux éléments successifs d'une table ou bloc de données. Le rôle de ces registres sera détaillé dans la leçon consacrée à l'adressage.

A noter toutefois que nous avons déjà utilisé le registre index IX dans le programme qui permettait de faire apparaître un message donné. Il s'agissait d'accéder à une table qui contenait les différents codes du message à visualiser (Led Micro n°9).

Le troisième registre 16 bits est le registre SP (Stack Pointer) ou Pointeur de Pile. La «pile» est une zone de la mémoire vive (donc exclusivement RAM), de taille essentiellement varia-

AF	BC	DE	HL	SP	IX	IY	PC
	01	11	21	31	DD	FD	
	n	n	n	n	21	21	
	n	n	n	n	n	n	
					n	n	

Fig. 31

ble allouée au stockage temporaire d'informations qui peuvent être soit des données soit des adresses. Par exemple, la «pile» est couramment employée pour la sauvegarde de l'adresse courante dans les opérations de «sauts de programme» (Réponse à une interruption par exemple).

Le quatrième et dernier registre est le compteur des programmes ou PC (Program Counter), on lui donne aussi le nom de Compteur Ordinal (CO). Il contient l'adresse de la prochaine instruction à exécuter.

Le tableau de la figure 31 indique les différents codes «opération» pour le chargement immédiat de données «16 bits».

Remarques

1) Il n'est pas possible de «charger» avec une donnée 16 bits la paire de registres A et F, ce qui confirme que les registres A et F bien qu'«associés» ne constituent jamais l'équivalence d'un registre 16 bits.

2) Il n'apparait pas d'instruction pour le chargement immédiat de PC. En fait l'instruction existe, (elle s'appelle CALL) mais elle n'appartient pas à la famille «Load».

3) Le chargement des registres IX et IY nécessitent quatre octets.

4) Pour procéder au chargement d'un registre double, dans l'écriture de l'instruction en langage machine, **l'octet de poids faible précède l'octet de poids fort.**

C'est une règle de syntaxe impérative.

Exemples :

Ld IX, 1808 s'écrit

DD 21 **08** **18**

Ld BC, 1234 s'écrit

01 **34** **12**

Vérifiez cette dernière remarque (importante) avec le MPF-1.

Avant d'étudier plus en détail le mécanisme de l'exécution d'un programme et, de ce fait, le rôle primordial que joue le compteur ordinal, nous indiquerons sur la figure 32, la configuration globale des registres du Z80.

III. DEROULEMENT D'UN PROGRAMME

III.1. Introduction

Nous savons que le programme à exécuter est enregistré dans la mémoire sous la forme d'une séquence d'instructions où chaque octet occupe une case mémoire. Ceci implique que le microprocesseur « connaisse » à chaque instant quelle est l'instruction à exécuter ou plutôt sache qu'elle est son adresse correspondante en mémoire. C'est précisément le rôle du PC que de contenir cette adresse.

Les instructions sont de longueur variable. Elles comportent au minimum un seul octet et donc n'occupe qu'un emplacement mémoire, et au maximum quatre octets, auquel cas l'instruction utilise quatre emplacements « mémoire » consécutifs. Ainsi, dire que le registre 16 bits PC pointe sur l'adresse à laquelle est stockée l'instruction, est vrai sans restriction quand celle-ci ne comporte qu'un seul octet (DAA par exemple). Par contre, quand l'instruction est constituée de 2, 3 ou 4 octets, **l'adresse est celle du premier octet.** Ceci reste homogène avec l'écriture d'un programme telle que nous l'avons fait jusqu'à présent puisque nous n'indiquons que l'adresse du premier octet et une seule instruction par ligne.

Comment le PC détermine-t-il la longueur d'une instruction ? La première étape dans l'exécution d'une instruction consiste à lire le premier byte de celle-ci et de le transférer dans le registre instruction qui appartient à l'Unité de Contrôle ou de Commande. Cette dernière possède un **circuit de décodage capable d'identifier à partir du premier octet la longueur totale de l'instruction.** Ainsi l'Unité de Contrôle déclenche le nombre de cycles de lecture nécessaire pour que **l'instruction toute entière** soit disponible dans le registre adéquat. Comme l'Unité de Contrôle incrémente automatiquement le PC après chaque lecture, quand l'instruction est entièrement chargée dans le CPU, **mais pas encore exécutée,** le compteur de programme pointe sur le début de l'instruction suivante. Ceci se vérifie aisément quand vous exécutez un programme en « pas à pas » avec le MPF-1.

En définitive, le PC n'a qu'un rôle de compteur qui est incrémenté après chaque cycle de lecture.

Cependant, dans certains cas, le

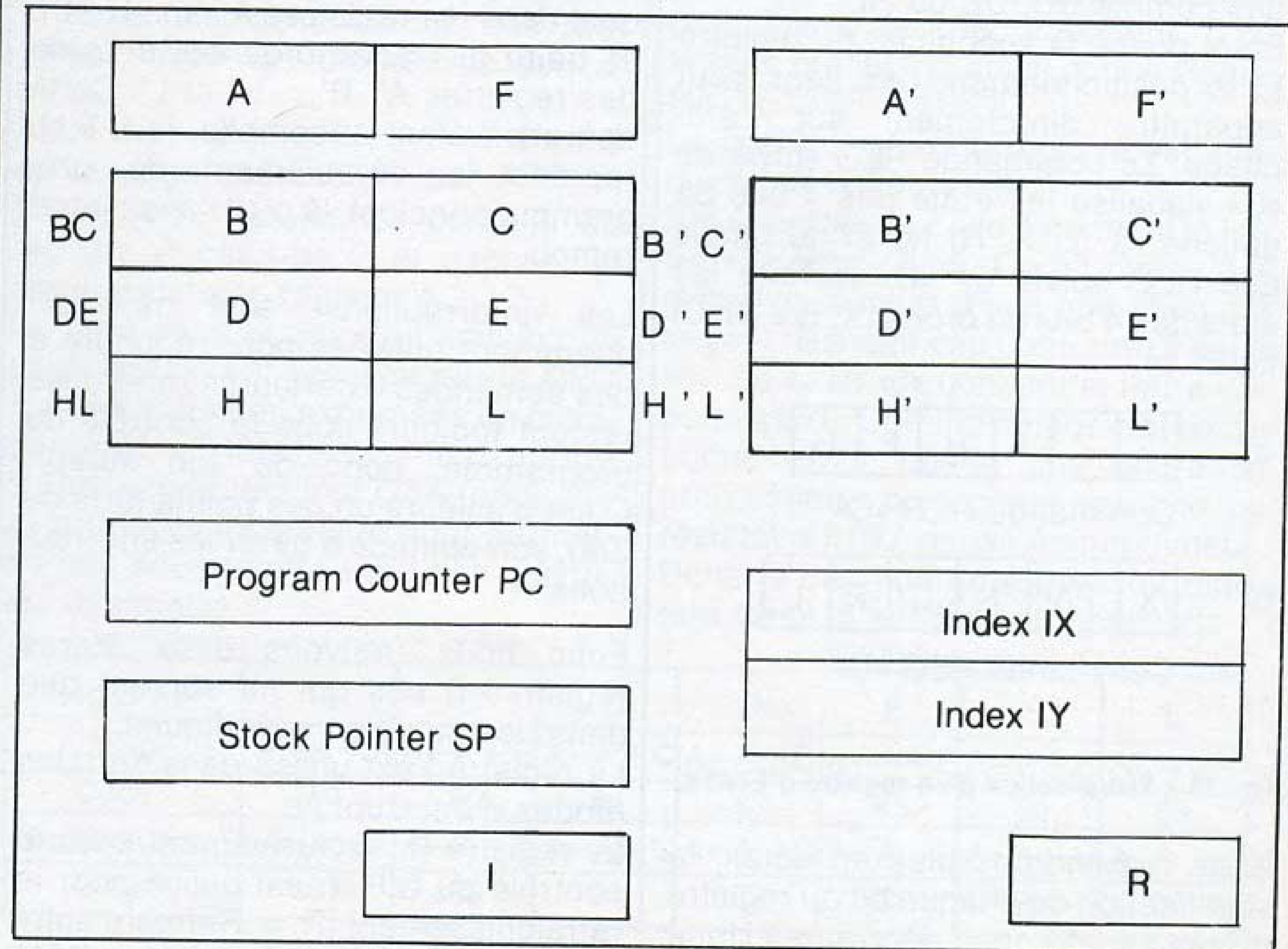


Fig. 32 : Configuration des registres.

contenu du PC peut être modifié par le programme lui-même. C'est-à-dire qu'il est possible d'exécuter des « déroutements » : dans le déroulement normal de la séquence pour exécuter des instructions stockées dans une autre zone de la mémoire. Ceci se réalise notamment avec des instructions de saut. Nous montrerons un exemple dans le cadre de cette étude.

Pour exécuter un programme, dès que celui-ci est introduit dans la mémoire, il suffit de « charger » le compteur PC avec l'adresse de départ puis de le lancer avec [GO] ou [STEP].

III.2. Exécution d'un programme

Le programme représente pour l'utilisateur une solution parmi d'autres, au problème posé. La tâche à accomplir est décomposée en une succession d'opérations que le microprocesseur peut identifier et exécuter.

La première étape pour mener à bien une tâche quelle qu'elle soit, et plus encore pour écrire un programme, est de bien définir ce que l'on désire réaliser.

Dans le cas présent, le programme consiste à additionner (à l'aide de l'accumulateur) le contenu du registre B et celui du registre C et de replacer le résultat, donc la somme, dans le registre C : opération qui peut se représenter symboliquement de la manière suivante :

$$C \leftarrow B + C$$

(la valeur « origine de C est écrasée par le résultat).

Pour fixer les idées, chargeons le registre B avec 14 H (la présence de H indique qu'il s'agit d'une quantité hexadécimale) soit 20 en décimal, et le registre C avec 05 H (05 en décimal). L'étude de l'UAL nous a montré qu'il pouvait réaliser (entre autre) l'addition du contenu de l'accumulateur (1er opérande) avec celui d'un autre registre désigné (2e opérande) et déposer le résultat (la somme dans le cas de l'addition) dans le registre A.

La figure 33 illustre les différentes liaisons qui assurent le cheminement des informations.

Comme il s'agit de quatre opérations que peut exécuter le microprocesseur, nous obtenons le programme suivant :

Adresse	Code Hexa	Instruction
(1) 1800 H	78	LD A,B
(2) 1801 H	81	ADD A,C
(3) 1802 H	4F	LD C,A
(4) 1803 H	76	Halt.

(A noter que le chargement des registres B et C peut être effectué avec la commande [REG] suivie de [DATA], comme indiqué dans le Manuel Technique du MPF-1, page 16).

L'adresse 1800 H et les suivantes indiquent les emplacements de la mémoire et leur contenu : les codes «opération» de notre programme. Sur la figure 34 nous avons utilisé la représentation «hexadécimale» et «binaire».

L'exécution du programme en «pas à pas» :

a) Le compteur de programme (PC) est chargé avec la première adresse du programme, dans notre exemple : 1800 H.

b) Le premier byte (78 H) est chargé dans le C.P.U.

c) Le compteur de programme est incrémenté de 1 (il contient 1801 H).

d) Le C.P.U. exécute l'instruction 78 H : le registre A est chargé avec le contenu du registre B. L'accumulateur contient 14 H, le registre B est

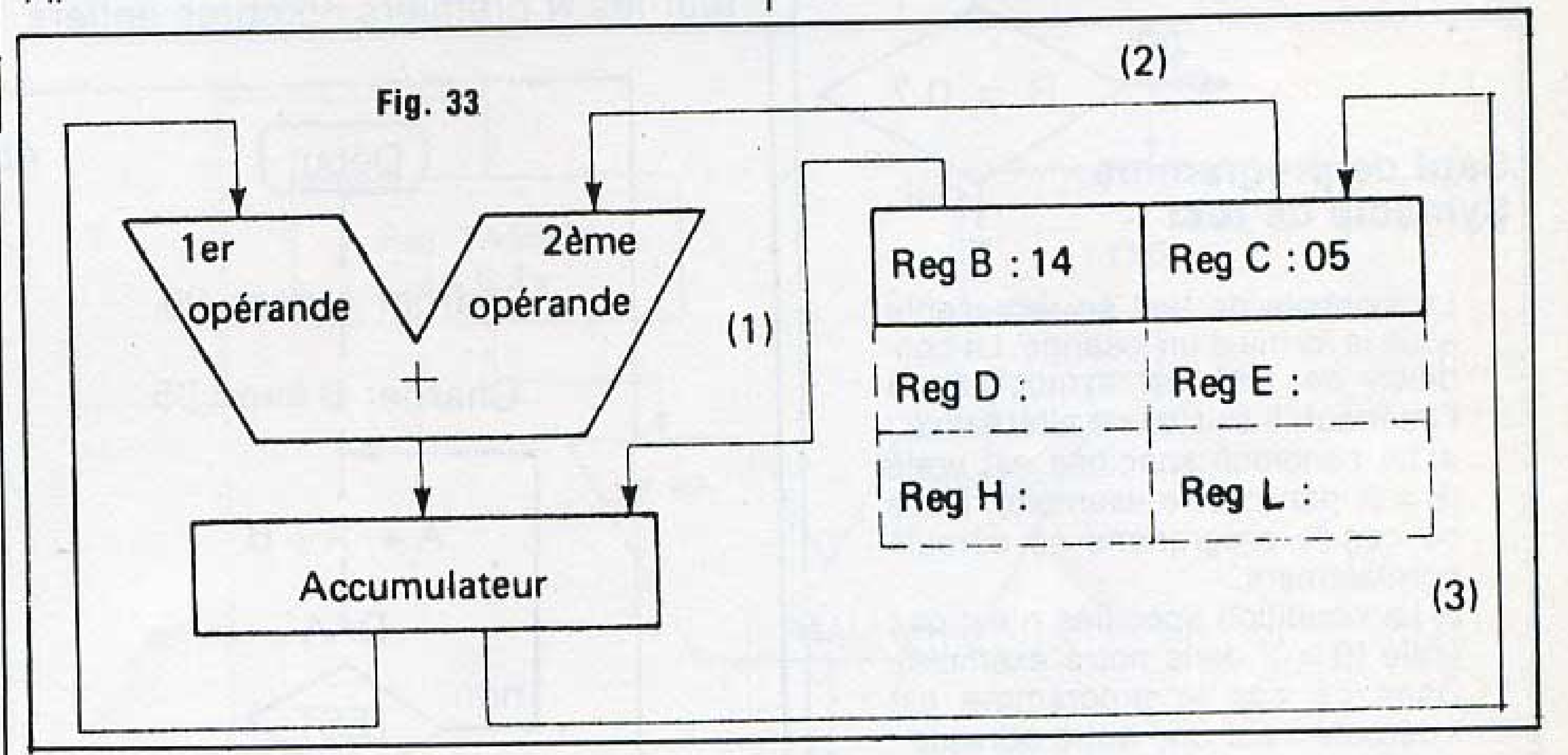


Fig. 33

Le séquencement des opérations est le suivant :

1. Charger le registre A avec le contenu du registre B (liaison 1).
2. Additionner le contenu de A (1er opérande) avec le contenu du registre C (liaison 2).
Déposer le résultat (A+C) dans l'accumulateur.
3. Charger le registre C (liaison 3) avec le contenu du registre A.
4. Arrêt. Fin de programme.

Fin de programme

• Quelques exemples de fin :

Code	Commentaires
76	Halt. Arrêt du programme La LED rouge d'allume
F7	RST 30. Retour au Moniteur
18 FE	JR, – 2. Boucle sur elle-même

• En n'indiquant pas au microprocesseur la fin d'un programme, celui-ci continue d'exécuter les instructions qu'il trouve en mémoire. Cette « exécution erratique » peut conduire non seulement à **une destruction du ou des programmes en mémoire RAM** mais aussi **à une modification des données :** ce qui rend inexploitable le travail en cours.

Dans la plupart des cas, l'instruction RST 30 (code F7) qui conduit à un Retour au Moniteur est la solution la plus adéquate.

Les deux autres instructions nécessitent un RESET pour redonner le contrôle au moniteur.

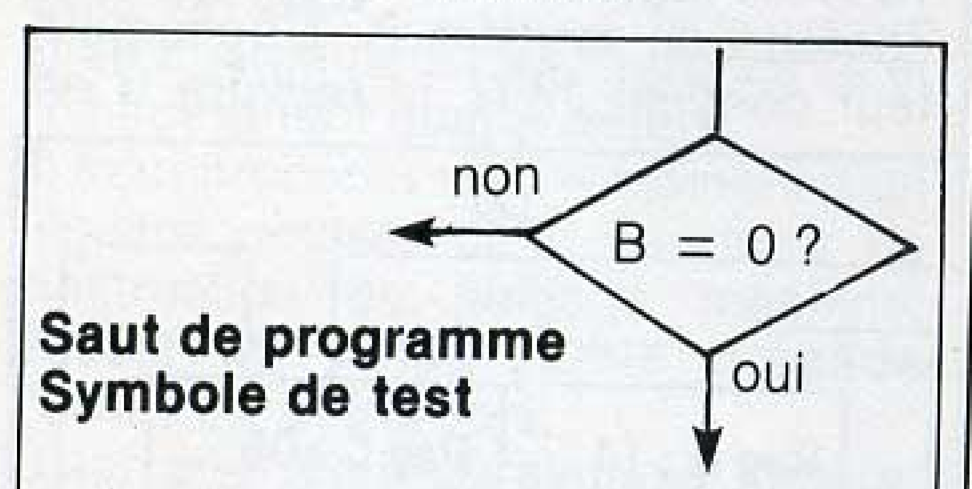
Principales commandes

Les numéros de page sont ceux du Manuel Technique du MPF-I-B

RS	Remise à zéro du système	p. 13
ADDR	Lecture en mémoire	p. 13
DATA	Ecriture en mémoire	p. 14
REG	Lecture et Ecriture d'un registre	p. 16
PC	Initialisation du C.O.	p. 19
GO	Exécution du programme	p. 21
STEP	Exécution en «pas à pas»	p. 21

inchangé. Instruction terminée.
e) Lecture de l'octet «pointé» par PC : l'octet 81 H (emplacement 1801 H) est chargé dans le C.P.U.
f) Le compteur de programme est incrémenté de 1 (il contient 1802 H).
g) Le C.P.U. exécute l'instruction 81 H : addition du contenu de A et du contenu de C. Résultat dans A. Instruction terminée.
h) Lecture de l'octet «pointé» par PC : l'octet 4 F H (emplacement 1802 H) est chargé dans le C.P.U.
i) Le compteur de programme est incrémenté de 1 (il contient 1803 H).
j) Le C.P.U. exécute l'instruction 4 F H : le contenu du registre A est recopié dans le registre C. Le contenu précédent (05 H) est écrasé par le nouveau résultat 19 H. A reste inchangé. Instruction terminée.
k) Lecture de l'octet «pointé» par PC : l'octet 76 H (emplacement 1803 H) est chargé dans le C.P.U.
l) Le compteur de programme est incrémenté de 1 (il contient 1804 H).
m) Le C.P.U. exécute l'instruction 76 H : Arrêt du système.

Notons bien que le P.C. indique toujours **l'adresse de la prochaine instruction.** Par exemple, si nous avons la possibilité de lire le contenu du PC après l'exécution du programme que nous venons de détailler, celui-ci n'affichera pas 1803, qui est la dernière adresse mais au contraire 1804 qui est la suivante. Par contre la **dernière instruction exécutée** est bien 1803 : arrêt du système.



Saut de programme
Symbole de test

Le symbole de test se représente sous la forme d'un losange. La condition de test est symbolisée à l'intérieur. Il existe une alternative :
a) La condition spécifiée est vraie (B = 0, dans notre exemple). Dans ce cas le programme se déroule normalement.
b) La condition spécifiée n'est pas vraie (B = 0, dans notre exemple). Dans ce cas le programme est « dérouté » sur une autre adresse :
• soit par addition d'un déplacement en valeur algébrique, comme dans le cas de l'instruction DJNZ ;
• soit en indiquant l'adresse à laquelle le programme doit se rendre.

(Adresse)	HEXA	BINAIRE
1800H	78	0111 1000
1801H	81	1000 0001
1802H	4F	0100 1111
1803H	76	0111 0110
1804H	18	0001 1000

Fig. 34

Nous venons de voir que le compteur de programme était augmenté de 1 après chaque cylce de lecture. Il existe cependant des instructions dont l'exécution modifie le contenu du PC.
Les instructions de «saut» seront examinées en détail par la suite, il est cependant important d'illustrer à l'aide d'un exemple, ce «déroutement» dans le séquencement d'un programme.
Nous avons choisi l'une des instructions de saut des plus performantes que nous utiliserons assez souvent dans les exemples ou les exercices qui viendront : c'est l'instruction DJNZ (abréviation de «Do Jump if No Zero»).
Pour illustrer cette nouvelle notion, utilisons un exemple :
Soit à additionner dans l'accumulateur les N premiers nombres entiers.

Prenons N = 5.
Nous pouvons proceder de la manière suivante :

$$A = 1 + 2 + 3 + 4 + 5 = 15$$

ou encore comme ceci :

$$A = 5 + 4 + 3 + 2 + 1 = 15$$

Nous adopterons cette deuxième possibilité. La méthode est la suivante : charger un registre (B par exemple) avec la valeur maximale (donc N = 5), additionner B au contenu de A A←A + B), retrancher une unité au contenu du registre B après chaque opération d'addition (B←B − 1). Répéter la séquence aussi longtemps que B n'égale pas zéro.
Quand B = 0, le programme est terminé.
C'est ce que traduit l'organigramme de la figure 35. Ce mode de représentation a pour but d'indiquer la

	Adresse	Code opératoire
Début		
Charger A avec 00	1800	3E 00
Charger B avec 05	1802	06 05
A ← A + B	1804	80
DAA	1805	27
TEST (non / oui)	1806	10 FC
Retour moniteur	1808	F7
FIN		

Fig. 35

séquence logique des opérations.
Nous avons présenté l'ensemble du programme à côté de l'organigramme, mise à part l'instruction de l'adresse 1806 (10 FC) celui-ci ne présente aucune difficulté.

Le contenu de l'adresse 1806 (10 FC) est précisément l'instruction «DJNZ» (code 10-d) dont les effets sont :

a) **diminue** de 1 le contenu du registre B, ou réalise l'opération B ← B−1.

b) **réalise le test suivant :**

— Si après l'opération a), le contenu du registre B est nul, le programme se déroule normalement. Comme «DJNZ» est une instruction 2 octets, après 1806, c'est l'instruction contenue en 1806 + 2 = 1808, donc «F7 (Retour moniteur)»qui est exécutée.

— Si après l'opération a), le contenu du registre B est différent de zéro, le programme effectue **un déplacement** ou **un «SAUT»** dont la valeur relative est l'octet qui suit l'instruction «DJNZ» (code 10).

Dans notre exemple, comme l'indique l'organigramme, nous devons nous rendre en 1804. Pendant l'exécution de DJNZ (2 octets), le registre PC contient 1808. Il faut donc lui retrancher 4 (1808 − 1804 = 4) ou lui ajouter − 4 (FC).

Rappel (encadré)

Nous reviendrons ultérieurement sur les «sauts de programme» ; notre objectif était de montrer dès à présent que le compteur de programme peut être modifié par certaines instructions.

Exécutez en «pas à pas» le programme de la figure 35. Observez tout particulièrement le contenu des registres A, B et le registre PC.

III.3. Registres «tampon»

Reprenons notre programme du paragraphe précédent qui consiste à additionner au contenu du registre B celui de C de déposer celui-ci dans C.

La première instruction notée en langage assembleur LD A, B n'appelle pas de commentaire particulier pour son exécution : le registre A est mis en communication avec B (liaison 1) et le transfert s'opère. Par contre la seconde, notée ADD A, C amène les remarques qui suivent.

Le contenu de l'accumulateur (14 H dans notre exemple) constitue le premier opérande. Le second est constitué par le contenu du registre C. **Le résultat doit apparaître dans A :** celui-ci joue donc pour cette opération **à la fois le rôle de la source** (1er opérande) et **de la destination** (le résultat). C'est dans ce double rôle que le problème apparaît (fig. 36).

Rappel :

en Hexad. :	F9	FA	FB	FC	FD	FE	FF	00
en décimal :	− 7	− 6	− 5	− 4	− 3	− 2	− 1	00

L'Unité Arithmétique et Logique est une combinaison de fonctions logiques classiques (ET, OU, etc.) nécessaires pour la réalisation d'additionneur, soustracteur, etc., comme ils sont expliqués dans les cours de Logique : la sortie est une fonction combinatoire des données d'entrée.

En chargeant le résultat de l'addition dans l'accumulateur, son contenu initial (14 H) qui est aussi le premier opérande est détruit puisqu'il est remplacé par la somme (19 H). Compte tenu de la liaison (Accumulateur-Entrée 1er opérande), **la quantité 19 H se substitue à celle d'origine 14 H,** ce qui conduit à un résultat différent dans l'accumulateur (19 H + 05 H). Ce dernier résultat va devenir à son tour le premier opérande et générera un autre résultat... et ainsi de suite le cycle peut se poursuivre.

Tel qu'il apparaît sur la figure 36, l'ensemble UAL plus Accumulateur risque de ne jamais fournir un résultat correct dans A : il faut introduire un «retard» entre la sortie et l'entrée

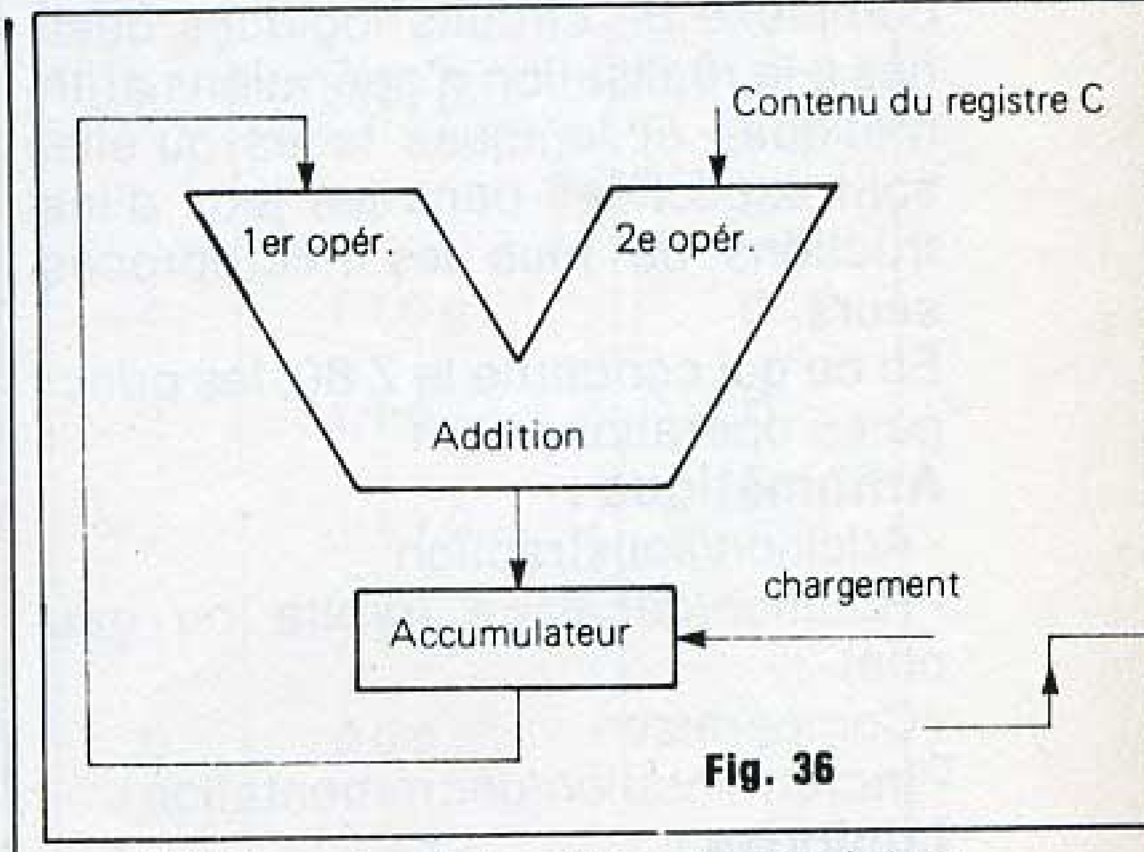


Fig. 36

de l'UAL : c'est le rôle du registre TAMPON dit encore Accumulateur Temporaire (ACT) qui apparaît sur la figure 37.

Par symétrie, la deuxième entrée est aussi précédée d'un registre tampon noté TMP.

Le lecteur doit se poser la question suivante : pourquoi n'avons-nous pas parlé de ces registres 8 bits dans l'étude précédente ?

En fait, ces deux nouveaux registres peuvent être ignorés par l'utilisateur. Ils sont «transparents» au programmeur et n'apparaissent dans aucune instruction. Cependant, il est bon de connaître leur existence, car beaucoup d'opérations portent sur **le contenu de l'accumulateur et... le résultat est replacé dans celui-ci.**

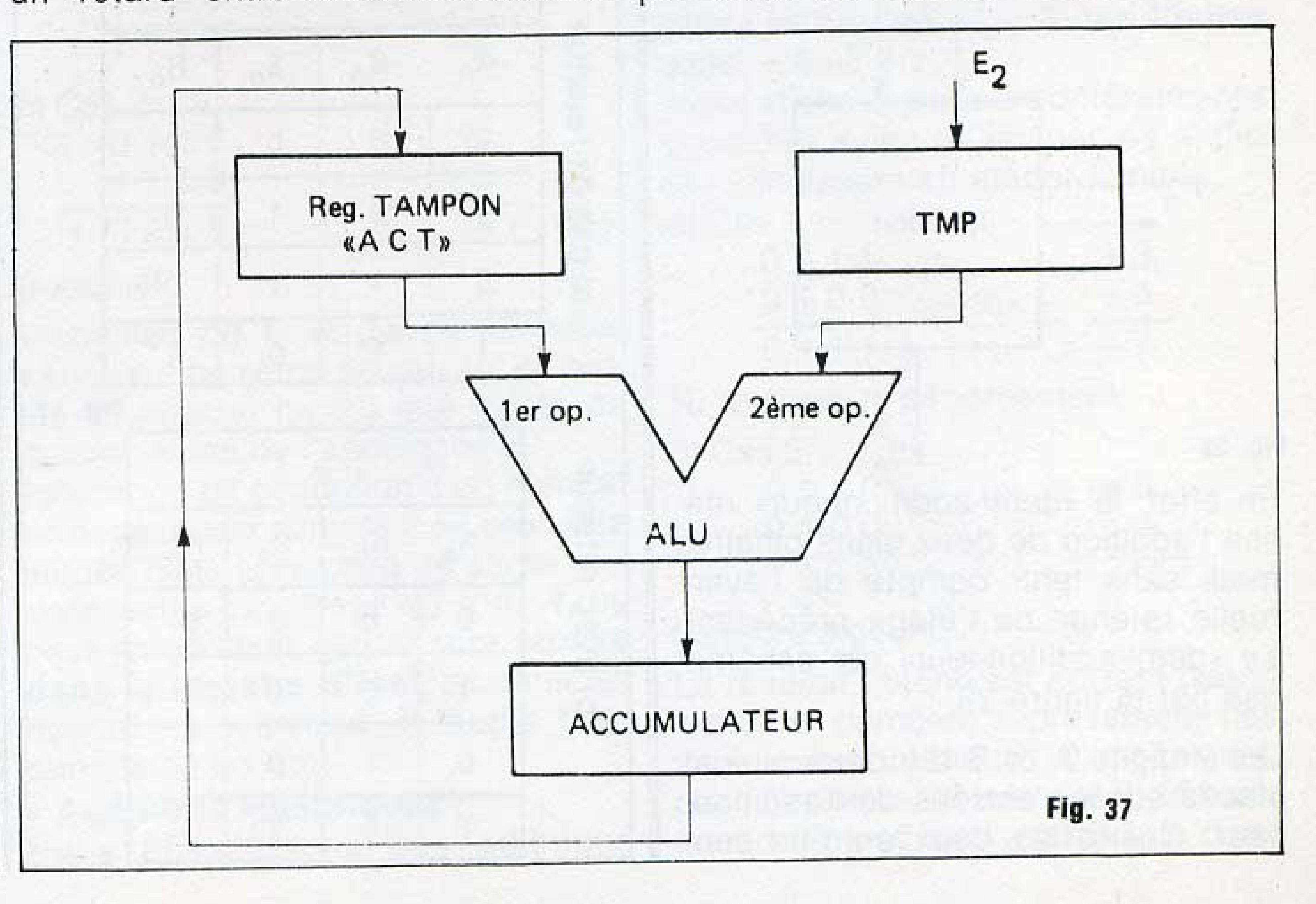


Fig. 37

IV. L'UNITE ARITHMETIQUE ET LOGIQUE

IV.1. Description

Le module Arithmétique et Logique (ALU) est un ensemble combinatoire complexe de circuits logiques destinés à la réalisation d'opérations arithmétiques et logiques telles qu'elles sont explicitées dans les jeux d'instructions de tous les microprocesseurs.

En ce qui concerne le Z 80, les principales opérations sont :

Arithmétique :
- Addition/soustraction
- Rotation/décalage (droite ou gauche)
- Comparaison
- Incrémentation/décrémentation

Logiques :
- ET, OU, OU Exclusif, Inversion
- Rotation/décalage (droite ou gauche)
- Positionnement et test d'un bit
- Complémentation.

Nous ne nous attarderons pas sur l'aspect hardware de ce module, mais nous nous intéresserons plus particulièrement aux opérations qu'il réalise et aux résultats obtenus, ainsi qu'au registre d'ETATS (ou registre F) qui est associé à l'ALU.

Succinctement, la partie logique de l'ALU peut être considérée comme un ensemble de circuits «additionneurs». Rappelons que chaque circuit, pris individuellement, est constitué lui-même de deux demi-additionneurs.

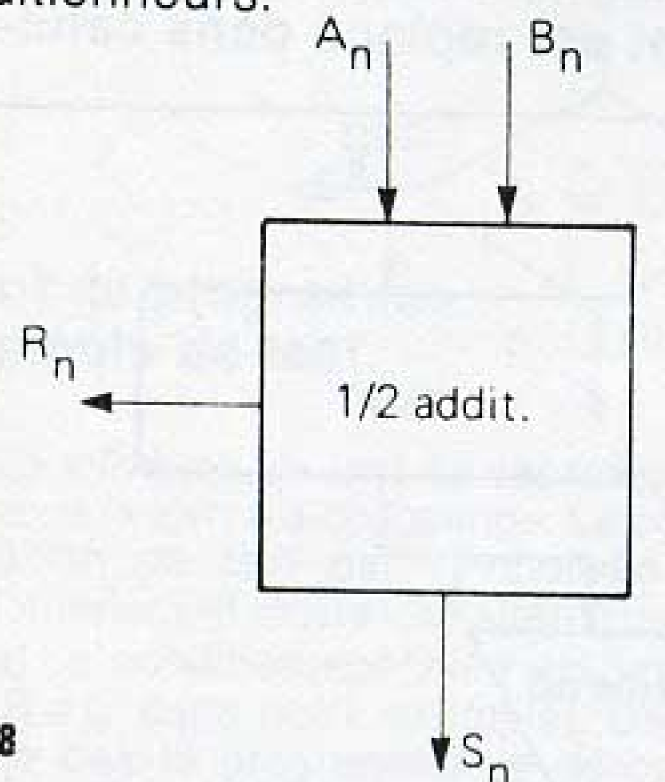


Fig. 38

En effet, le «demi-additionneur» réalise l'addition de deux digits binaires mais sans tenir compte de l'éventuelle retenue de l'étage précédent. Le «demi-additionneur» est schématisé par la figure 38.

Les 2 digits A et B d'ordre «n» sont placés sur les entrées de l'additionneur. Quand les deux entrées sont simultanément à «1» une retenue $R_n = 1$ apparaît (table de vérité, figure 39. C'est cette retenue qui doit être additionnée dans l'étage suivant d'ordre «n + 1».

A_n	B_n	S_n	R_n
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Fig. 39

L'additionneur complet (en réalité 2 demi-additionneurs) tient compte de la retenue de l'étage précédent. Il reçoit (figure 40) sur des 2 entrées les deux digits A_n et B_n à additionner mais en plus la retenue R_{n-1} de l'étage précédent.

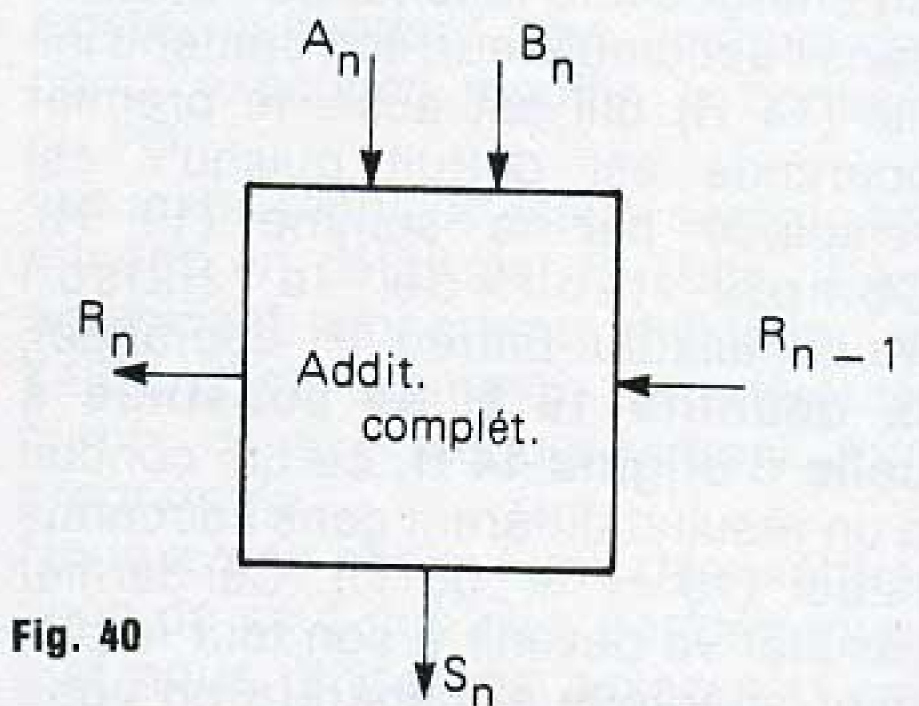


Fig. 40

Il fournit en sortie d'une part S_n représentant le bit somme d'ordre «n» et d'autre part la retenue R_n, dont les valeurs sont indiquées dans la table de vérité de la figure 41.

a) pas de retenue — $R_{n-1} = 0$

A_n	B_n	S_n	R_n
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

b) avec retenue — $R_{n-1} = 1$

A_n	B_n	S_n	R_n
0	0	1	0
1	0	0	1
0	1	0	1
1	1	1	1

Fig. 41

IV.2. Représentation des nombres

• Nombres représentés par un octet

L'accumulateur est un registre 8 bits, il peut donc réaliser des opérations dans lesquelles les données d'entrée et/ou de sortie peuvent se représenter par 1 octet.

Il faut distinguer deux cas suivant que le nombre représenté est un nombre arithmétique (ou sans signe) ou un nombre algébrique (ou avec signe).

• Nombres arithmétiques :

L'ensemble E des nombres entiers arithmétiques qui peuvent être représentés avec un mot de n bits est :

$$0 \leqslant E \leqslant 2^n - 1$$

Exemple :

si $n = 3$

l'ensemble E est

$$0 \leqslant E \leqslant 2^3 - 1$$
$$0 \leqslant E \leqslant 8 - 1 = 7$$

Et l'ensemble E des nombres arithmétiques qui peut être représenté par un octet (8 bits) est 0 pour le minimum et 255 ($2^8 - 1$) pour le maximum.

Tandis que l'ensemble E des nombres arithmétiques qui peut être représenté par deux octets (16 bits) est 0 pour le minimum et 65 535 ($2^{16} - 1$) pour le maximum.

• Nombres algébriques :

Conventionnellement, dans la représentation binaire des nombres algébriques, **le bit le plus significatif indique le signe.**

Ainsi, les **nombres positifs ont leur bit le plus significatif à «0»,** tandis que les **nombres négatifs ont leur bit le plus significatif à «1».**

L'ensemble E des nombres entiers algébriques qui peuvent être représentés avec un mot de n bits est :

$$-2^{n-1} \leqslant E \leqslant 2^{n-1} - 1$$

Exemple :

si $n = 3$

l'ensemble E est

$$-2^{3-1} \leqslant E \leqslant 2^{3-1} - 1$$
$$\text{ou } -2^2 \leqslant E \leqslant 2^2 - 1$$

soit − 4, − 3, − 2, − 1, 0, 1, 2 et 3.

Et l'ensemble E des nombres algébriques qui peut être représenté par un octet (8 bits) est − 128 (-2^7) pour le minimum et + 127 ($+2^7 - 1$) pour le maximum.

A titre d'exemple, nous donnons la

représentation binaire, décimale et hexadécimale des nombres arithmétiques et algébriques dans le cas où n = 4 (voir les tableaux de la figure 42.

• Complément à 1 :
Cette opération réalisée par l'ALU est aussi connue sous le nom de fonction «complémentation» ou, en logique, de fonction inversion. L'opération consiste à remplacer dans un mot, chaque bit par sa valeur opposée. Les «1» sont remplacés par des «0» et les «0» par des «1».
Soit à calculer le complément de $(F3)_H$.

$$(F3)_H = 1111\ 0011$$
$$\overline{(F3)_H} = 0000\ 1100 = (OC)_H$$

l'inverse de $(F3)_H$ représenté par $\overline{(F3)_H}$ est $(OC)_H$.

• Complément à 2 :
Cette opération permet d'obtenir la **valeur algébrique opposée** d'une valeur signée.
Le complément à 2 d'une valeur binaire s'obtient en ajoutant 1 au «complément à 1» de cette valeur. Dans le cas où un bit de «retenue» apparaît, celui-ci est ignoré.
Exemple :
Soit à trouver le complément à 2 de 3 :

La valeur de 3 (quand n = 4) est : 0011
Le complément à 1 (on inverse) est : 1100
d'où complément à 2 est :

```
 1100
+   1
-----
 1101
```

La valeur binaire de + 3 est : 0011 (+ 3)
La valeur binaire de − 3, ou complément à 2 de + 3 est : 1101 (− 3)
conformément à ce qui est indiqué sur le tableau de la figure 42.

• Code BCD
La représentation BCD (Binary Coded Decimal) est une représentation mixte dans laquelle chaque digit décimal est remplacé par son équivalent binaire sur 4 bits ou 1 quartet.
Exemple :

$$(57)d = (0101\ 0111)_{BCD}$$

IV.3. Additions

• Additions arithmétiques :
Nous effectuerons ces additions,

Nombres «Arithmétiques»		
D	**Binaire**	**Hexa**
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Nombres «Algébriques»		
D	**Binaire**	**Hexa**
−8	1000	8
−7	1001	9
−6	1010	A
−5	1011	B
−4	1100	C
−3	1101	D
−2	1110	E
−1	1111	F
0	0000	0
+ 1	0001	1
+ 2	0010	2
+ 3	0011	3
+ 4	0100	4
+ 5	0101	5
+ 6	0110	6
+ 7	0111	7

bit de signe

Fig. 42

dans l'hypothèse où n = 4, c'est-à-dire avec des opérateurs binaires (données et résultats) de 4 bits.
a) Cas 1

```
  0101        5
  1001        9
  ----       --
  1110       14
```

b) Cas 2

```
  0110        6
  1100       12
 -----       --
 10010       18 (  15)
```

(retenue)
L'addition de 6 et 12 donne naissance à cinquième bit, qui est en réalité la retenue quand elle existe du dernier étage de l'additionneur.
Ignorer ce bit conduirait à un résultat erroné, aussi comme il ne peut être stocké dans le registre de sortie (qui dans notre hypothèse où n = 4, ne peut en contenir que 4) sera **stocké dans le registre d'état,** et dans les opérations suivantes, il faudra tenir compte de ce «bit».

• Additions algébriques :
Nous effectuerons ces additions dans l'hypothèse où n = 4.
Dans le cas de **nombres algébriques,** lorsque le résultat d'une addition donne un résultat «hors des limites» on dit qu'il y a «débordement» ou «overflow».

Dans le cas où n = 4, les limites sont − 8 et + 7.

Nous allons illustrer les différents cas possibles avant de donner les règles qui conduisent au «débordement».
a) Cas 1

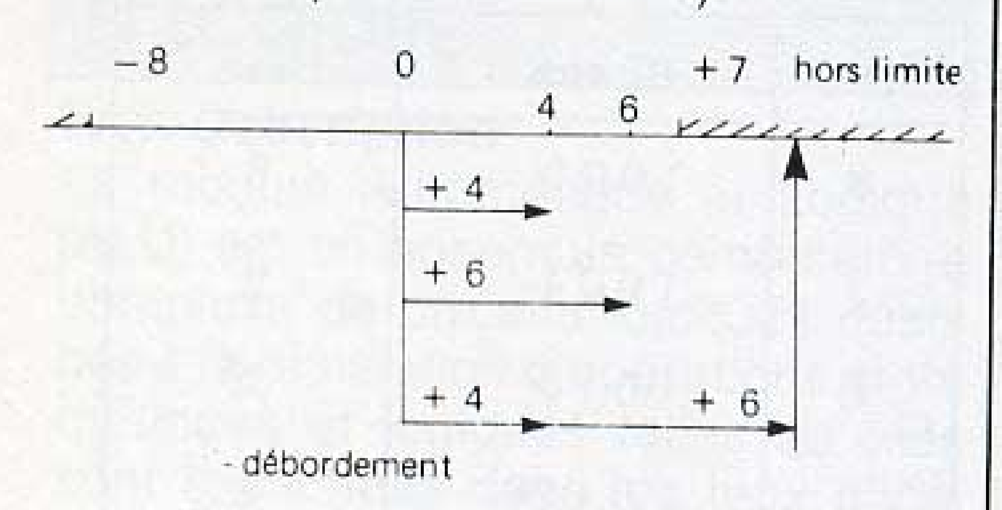
```
  0011        +3
  0100        +4
  ----        --
  0111 =      +7
```

Ni retenue, ni débordement
b) Cas 2

```
  0110        +6
  0100        +4
  ----        ---
  1010 =      +10
```

(nombre algébrique : − 6)
Le résultat obtenu est correct quand les deux nombres représentent des valeurs arithmétiques (sans signe), par contre le résultat est inexact quand il s'agit de l'addition de deux

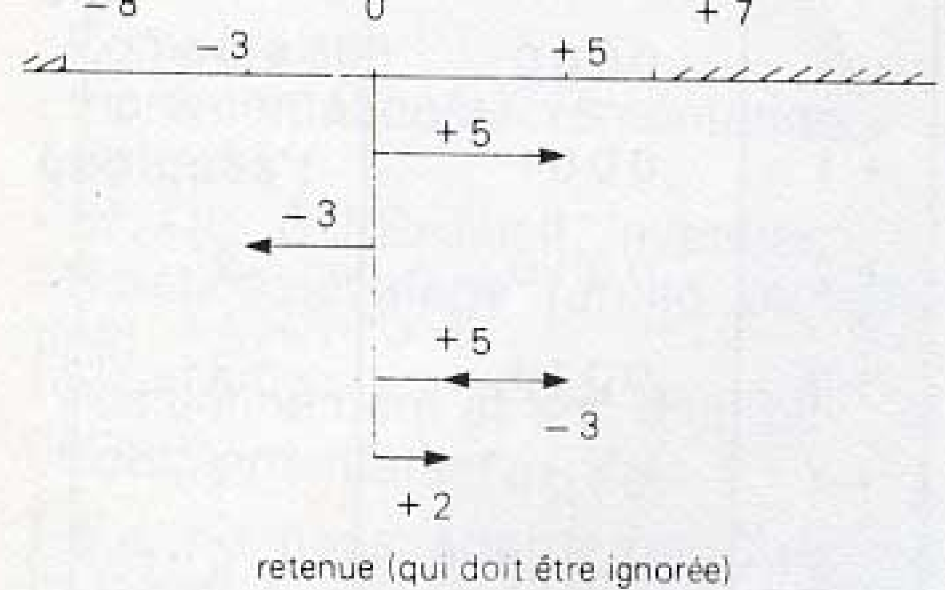
nombres algébriques (puisqu'on obtient l'équivalent de – 6).



c) Cas 3

```
            0 1 0 1       + 5
            1 1 0 1       – 3
(retenue :) 1 0 0 1 0  =  + 2
```

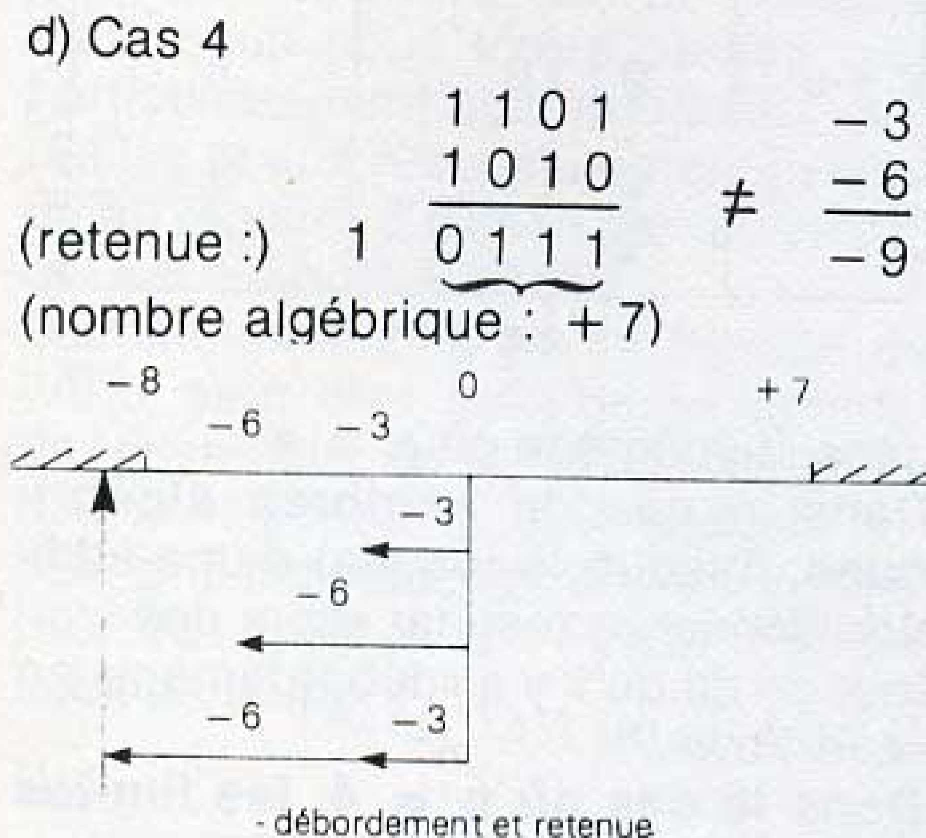
(nombre algébrique : + 2)



d) Cas 4

```
            1 1 0 1       – 3
            1 0 1 0   ≠   – 6
(retenue :) 1 0 1 1 1     – 9
```

(nombre algébrique : + 7)



En conclusion, les règles qui permettent de déterminer les conditions dans lesquelles le résultat de l'addition (dans le cas de nombres algébriques) sera «hors limites» (positives ou négatives) sont les suivantes :

1. le débordement (overflow) ou résultat «hors limites» n'apparaît que dans le cas de nombres algébriques
2. Lorsque les 2 nombres ont des signes opposés, le débordement est impossible (cas 3)
3. quand les deux nombres ont même signe, le débordement est possible et apparaît **si et seulement si le bit signe du résultat** (bit le plus significatif, mais qu'il ne faut pas confondre avec la retenue) **est l'opposé de celui des opérandes.**

Ainsi pour le cas 2 :
les 2 nombres à additionner sont positifs **0** 1 1 0 et **0** 1 0 0
et le résultat est négatif **1** 0 1 0
les 2 nombres ont même signe, le résultat est de signe opposé, il y a donc débordement.
dans le cas 4 :
nombres négatifs : **1** 1 0 1 et **1** 0 1 0
résultat positif : **0** 1 1 1

Nota : Dans le cas **d'additions de nombres algébriques,** le bit de «retenue» doit être ignoré.

Conclusions sur les Additions

Le microprocesseur exécute les opérations d'addition en utilisant des valeurs binaires, comme rappelé au début de ce paragraphe. Il **ignore** s'il s'agit de quantités arithmétiques (sans signe) ou algébrique (le bit de poids fort étant le bit de signe). Quoi qu'il en soit il positionne les indicateurs C et P/V dans tous les cas.

Par contre le programmateur, **sait,** si les quantités représentent des données arithmétiques ou algébriques et c'est à lui de tenir compte dans son programme :

- de l'indicateur C (report) quand il s'agit de valeurs arithmétiques et d'ignorer P/V ;
- de l'indicateur P/V (débordement) quand il s'agit de valeurs algébriques.

Le Z 80 R dispose de quatre autres indicateurs en plus de C et P/V qui sont regroupés dans le registre F.

IV.4. Registres d'états

Les registres d'ETATS (registre principal F et registre auxiliaire F'), associés à l'Unité Arithmétique et Logique, mémorise des «ETATS» particuliers résultant d'opérations arithmétiques ou logiques dans l'ALU à un moment donné.

Bien que les registres F soient des registres 8 bits, seuls 6 d'entre eux sont utilisés. **Chacun des 6 emplacements représente un indicateur,** indépendants les uns des autres et qui peuvent être testés par le programme (à l'exception des indicateurs N et H qui sont testés par une instruction particulière).

La figure 40 montre le format de ce registre particulier.

d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0
S	Z	X	H	X	P/V	N	C

X : non utilisé

a) Bit «C» : Retenue (CARRY) :

Le bit «C» de retenue du registre d'états est mis à 1 (C = 1) pour indiquer une retenue après une opération arithmétique (addition ou soustraction) et est mis à zéro (C = 0) s'il n'y a pas de retenue.

En addition, C est mis à 1, quand un report s'échappe du bit le plus significatif (bit 7 de l'accumulateur ou bit 15 de HL).

En soustraction, C est mis en 1, quand aucun report ne s'échappe du bit le plus significatif (bit 7 de l'accumulateur ou bit 15 de HL).

Exemples :

```
1)      0010  1111
      + 0110  1101
    [0] 1001  1100
    → C = 0

2)      1010  0011
      + 1101  1100
    [1] 0111  1111
    → C = 1

3)   0011  1101      61          0011  1101
   – 0011  0000    – 48        + 1101  0000
                     13      [1] 0000  1101  =  13
```

Nota : Une soustraction binaire peut toujours être remplacée par l'addition du complément à 2 du nombre à retrancher, ce qui revient à ajouter la valeur opposée.

```
                               → C = 0
4)   0001  1011      27          0001  1011
   – 0010  1011    – 43        + 1101  0101
                   – 16      [0] 1111  0000  ≠  240
                               → C = 1
```

Nota : Calculons la valeur complément à 2 du résultat obtenu : la valeur complémentée est :

```
               00001111
ajoutons 1 +          1
               --------
               00010000   = 16
```

Ainsi nous retrouvons bien le résultat de l'opération, après une action corrective : recherche de la valeur complément à 2.

Cet emplacement mémoire est aussi utilisé comme bit supplémentaire dans les opérations de décalage ou rotation que nous étudierons dans une leçon ultérieure.

b) Bit «Z» pour ZERO :

Le bit «Z» est mis à 1 quand le résultat d'une opération arithmétique ou logique donne un résultat nul (valable pour les opérations sur 1 ou 2 octets).

c) Bit «S» pour SIGNE :

Le bit «S» est l'image du bit de signe pour les nombres algébriques ou nombres signés. C'est le bit 7 quand ceux-ci sont représentés par un octet, c'est le bit 15 quand il s'agit de valeurs sur deux octets. Le bit «S» est 0 quand le nombre est positif, 1 quand il est négatif.

d) Bit «P/V» pour Parité d'une part ou Valeur hors gamme

Le bit «P/V» a une double signification.

1) Parité

Après une opération logique, telles que ET, OU, Rotation, etc., ce bit est mis à «1» quand la parité est paire (si le nombre de bits à 1 est pair) et il est mis à «0» si la parité est impaire.

Exemples :

```
      1001  1101
ET    0111  0101      | P/V = 1 |
      ----------
      0001  0101
          3 bits à 1 Parité impaire

      1001  1100
OU    0101  0101      | P/V = 0 |
      ----------
      1101  1101
          6 bits à 1 Parité paire
```

2) Valeur hors gamme :

Le résultat d'une opération sur des nombres algébriques peut conduire à des valeurs hors des limites de la gamme. Dans le cas d'une représentation par 1 octet, les limites décimales sont – 128, + 127.

Un débordement par valeurs positives (< + 127) est un «Overflow» tandis que le débordement par valeurs négatives (> – 128) est un «Underflow» : dans les deux cas, le bit «P/V» est positionné en 1.

Nota : Le débordement (bit «P/V») ne doit pas être confondu avec le dépassement de capacité (bit «C») comme le montrent les exemples ci-après.

Exemple 1 :

```
   0101  0101      85
   0111  0011 +   115
   -----------    ---
0  1100  1000 ≠   200  = (– 56 Faux)
```

Après l'exécution de cette instruction :

P/V = 1
C = 0

La condition de débordement est vraie : les signes des opérandes («0» donc positifs) et celui du résultat («1» donc négatif) sont opposé : c'est un cas de débordement donc bit «P/V» = 1.

Exemple 2 :

```
     1100  1000     – 56
 +   1101  1101     – 35
     ----------     ----
[1]  1010  0101  =  – 91
```

Après l'exécution de cette instruction :

P/V = 0

C = 1

La condition de débordement est vraie : les signes des opérandes («0» donc positifs) et celui du résultat («1» dont négatif) sont opposés : c'est un cas de débordement dont bit «P/V» = 1.

Nota :

Le complément à 2 de 1010 0101 est : 0101 1011 = 91.

La condition de débordement n'est pas vraie : les signes des opérandes et du résultat sont tous à 1, donc négatifs.

Exemple 3 :

Soit à additionner :

```
     1100  1000   (A)
 +   1000  0110   (B)
     ----------
[1]  0100  1110
```

Après l'exécution de cette instruction :

P/V = 1

C = 1

Puisque les bits des opérandes et du résultat sont opposés, il y a un report du dernier bit.

Dans cet exemple, nous n'avons pas précisé si les données étaient des valeurs arithmétiques ou algébriques.

Si (A) et (B) représentent des valeurs arithmétiques A = 200 ; B = 134, le résultat est 334, supérieur à 255, ce qui explique le report C = 1.

Si (A) et (B) représentent des valeurs algébriques A = – 122, B = – 56, le résultat est – 178, donc hors de la gamme, ce qui s'explique par le débordement P/V = 1.

e) Bit «H» ou HALF CARRY :

Le bit «H» ou demi-retenue (half-carry) est mis à 1 (H = 1) lors de l'exécution d'opération arithmétique qui ne porte que sur un seul octet, lorsqu'il y a une retenue du demi-octet de poids faible sur celui de poids fort. Il est remis à 0 (H = 0) dans le cas contraire (pas de retenue issue du demi-octet de poids faible).

Cet indicateur permet de réaliser l'addition ou la soustraction de nombres décimaux DAA, (Decimal Adjust Accumulator) il permet de corriger le résultat d'une opération arithmétique effectuée en BCD).

f) Bit «N» ou NEGATIF :

Ce bit «N» est utilisé pour identifier la dernière opération effectuée :

N = 0, si la dernière opération est une addition

N = 1, si la dernière opération est une soustraction.

L'indicateur «N» et l'indicateur «H» sont associés avec l'instruction DAA, pour effectuer la correction d'ajustement décimal dans le cas des opérations décimales codées en binaire.

Les autres indicateurs «Carry», «Zéro», P/V et Signe peuvent être testés individuellement par le programme et constituer ainsi une condition de branchement : un saut de programme qui ne dépend que du bit testé qu'il soit «1» ou «0» suivant la condition choisie (les branchements conditionnels feront l'objet d'une étude ultérieure).

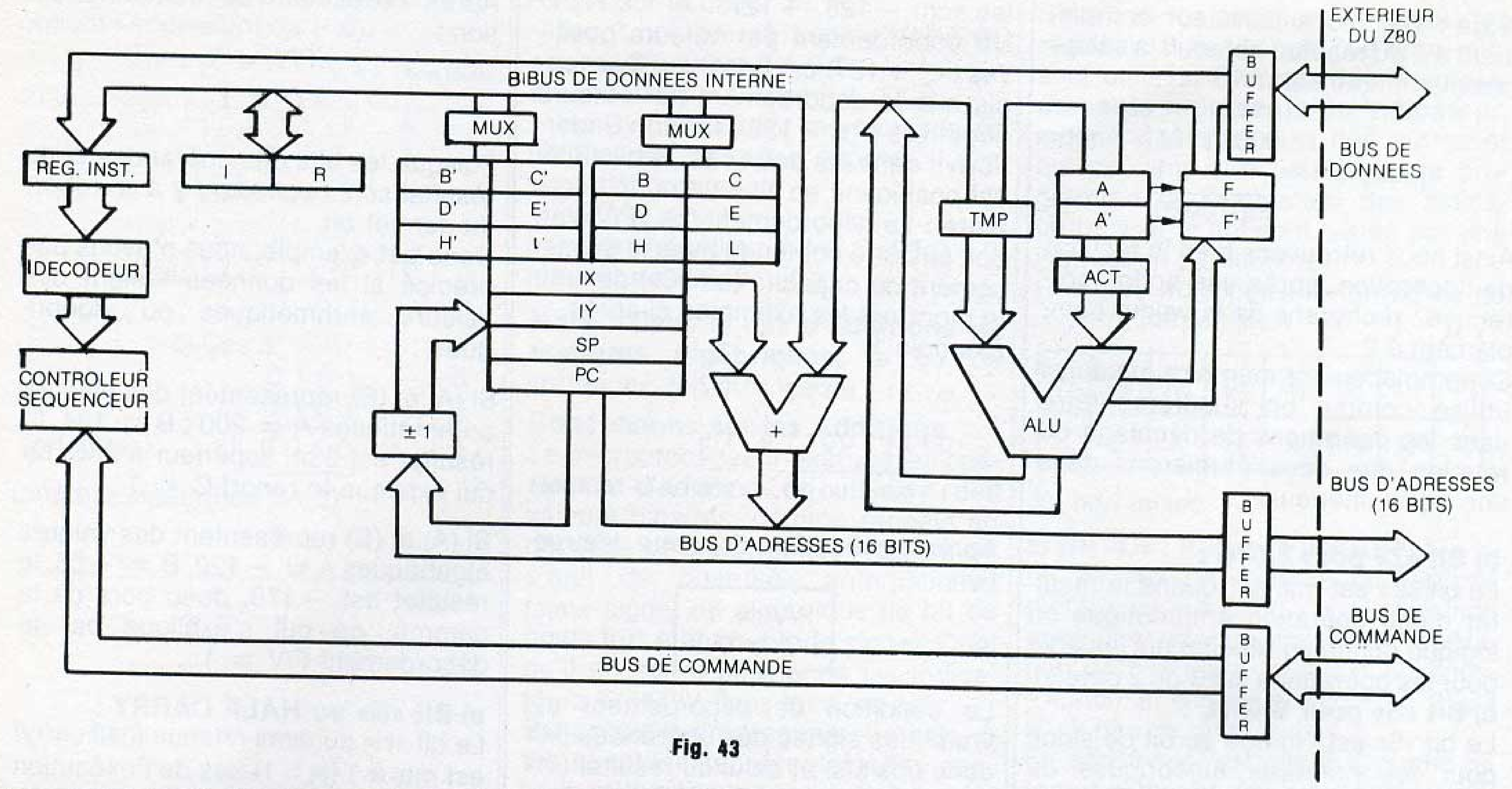


Fig. 43

V. ORGANISATION DU Z-80R

Les différents éléments, les plus importants, du Z 80R ont ainsi été présentés, ceux-ci peuvent être regroupés sous forme d'un synoptique (extrait de «Programmation du Z 80, de R. Zaks) donné par la figure 41.

On note que l'Unité de Contrôle ou Unité de Commande comporte plusieurs circuits, à savoir :

- le contrôleur de séquence
- le registre d'instruction
- le décodeur.

C'est ce dernier qui détermine la longueur de l'instruction mais aussi élabore les différentes commandes qui configurent l'UAL et génère les signaux de contrôle internes mais aussi externes, qui permettent le déroulement correct des instructions.

On note enfin que les trois bus, Données, Adresses et Contrôle, communiquent avec l'extérieur au travers d'amplificateurs de puissance appelés BUFFERS.

Dans notre prochain numéro, en étudiant l'architecture du MPF-1B nous replacerons le microprocesseur dans un environnement complet et réel. Nous aborderons ainsi les instructions qui permettent des échanges avec l'extérieur.

□

EXERCICE I

Quel intérêt peut présenter l'instruction dont le code est 97 ? Peut-on obtenir un résultat identique différemment ?

□

EXERCICE II

Réaliser un programme qui multiplie :
a) le contenu de A par 2, résultat dans A
b) le contenu de A par 4, résultat dans A
c) le contenu de A par 3, résultat dans A
(Ne pas utiliser DJNZ dans cet exercice).
Le résultat est exprimé en BCD dans A.

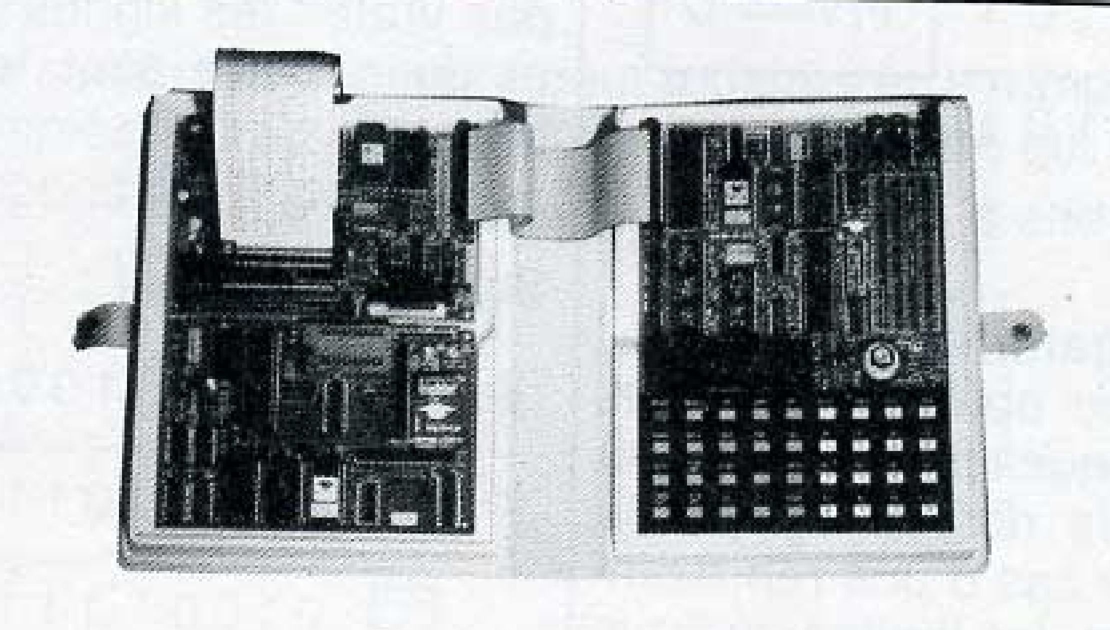
□

EXERCICE III

a) A l'aide des instructions Load et ADD (ainsi que DAA) uniquement, écrire un programme qui réalise la multiplication par 5 du contenu du registre C et dépose le résultat dans C. Le nombre N initial est perdu.
Avant exécution, C contient N
Après exécution, C contient 5 × N.
Pour bénéficier d'un moyen de contrôle aisé, effectuez le programme sur des quantités décimales avec N<20.

b) Réaliser un programme identique utilisant l'instruction DJNZ. Comparez avec III.a.

c) Pour réaliser une multiplication par 3 ou 7, quelle solution a) ou b) vous semble la plus adéquate. Pourquoi ?



COURS PRATIQUE DE MICROPROCESSEUR AVEC LE MICROPROFESSOR MPF-IB

TROISIEME PARTIE

Le hardware du MPF-IB (I)

SOMMAIRE

Solution de l'exercice 2 dans le numéro 12

I. INTRODUCTION

Dans la première partie du cours (L.M. n° 9) nous avons guidé le lecteur dans la découverte du microprocesseur ainsi que des circuits nécessaires qui constituent un système informatique. Dans la deuxième partie, nous avons présenté les différents éléments ou blocs fonctionnels que renferme le microprocesseur. Nous allons terminer cet aspect matériel, en étudiant dans ce numéro et le suivant, l'environnement extérieur, toujours à l'aide d'un système complet et réel le Microprofessor MPF-1B.

Ainsi c'est avec une bonne connaissance « hardware » que nous aborderons ensuite le « langage du microprocesseur », c'est-à-dire le logiciel.

II. LE MICROPROCESSEUR ET SON ENVIRONNEMENT

1. Le boîtier

Le microprocesseur par lui-même est réalisé sur une puce de silicium de 4,54 mm par 4,87 mm ce qui représente une surface de 22,22 mm², encapsulée dans un boîtier standard «dual in line» de 40 broches. Les connections entre la puce et les sorties du boîtier sont effectuées par du fil d'or très fin, au cours de l'opération de «bounding».

Le type de boîtier choisi est avant tout dicté par des raisons économiques. Habituellement pour les microprocesseurs 8 bits, les constructeurs s'en tiennent aux standards 40 ou 42 broches (ce qui ne va pas sans poser un certain nombre de problèmes). Les stations de tests, entièrement automatiques et pilotées par ordinateur qui contrôlent les «puces» et les circuits finis, n'acceptent presque jamais des boîtiers de plus de 42 broches et une station coûte plusieurs millions de francs...

2. Brochage du Z 80 :

La figure 44 donne la configuration des broches du Z 80 :

Ainsi, trois groupes de signaux apparaissent :

Le premier groupe (en haut, à droite) nous est presque familier, il s'agit des 16 sorties du Bus d'Adresses (16 connections).

Le second groupe (en bas, à droite) ne nous est pas inconnu, c'est le bus de données qui comporte 8 lignes (8 connections).

Le troisième groupe (à gauche) rassemble en trois sous-groupes l'ensemble des signaux de commande du CPU (6, 5, 2, soit 13 connections).

Ces trois groupes totalisent ainsi 37 connections, il nous en reste 3 qui n'entrent apparemment dans aucun des autres groupes et que nous allons étudier tout de suite.

Les broches 11 et 29, respectivement notées + 5 V et masse sont utilisées pour l'alimentation du circuit. Notons que ces deux broches ne se trouvent pas aux emplacements stratégiques (7 et 14, ou 8 et 16) comme nous avons coutûme de les rencon-

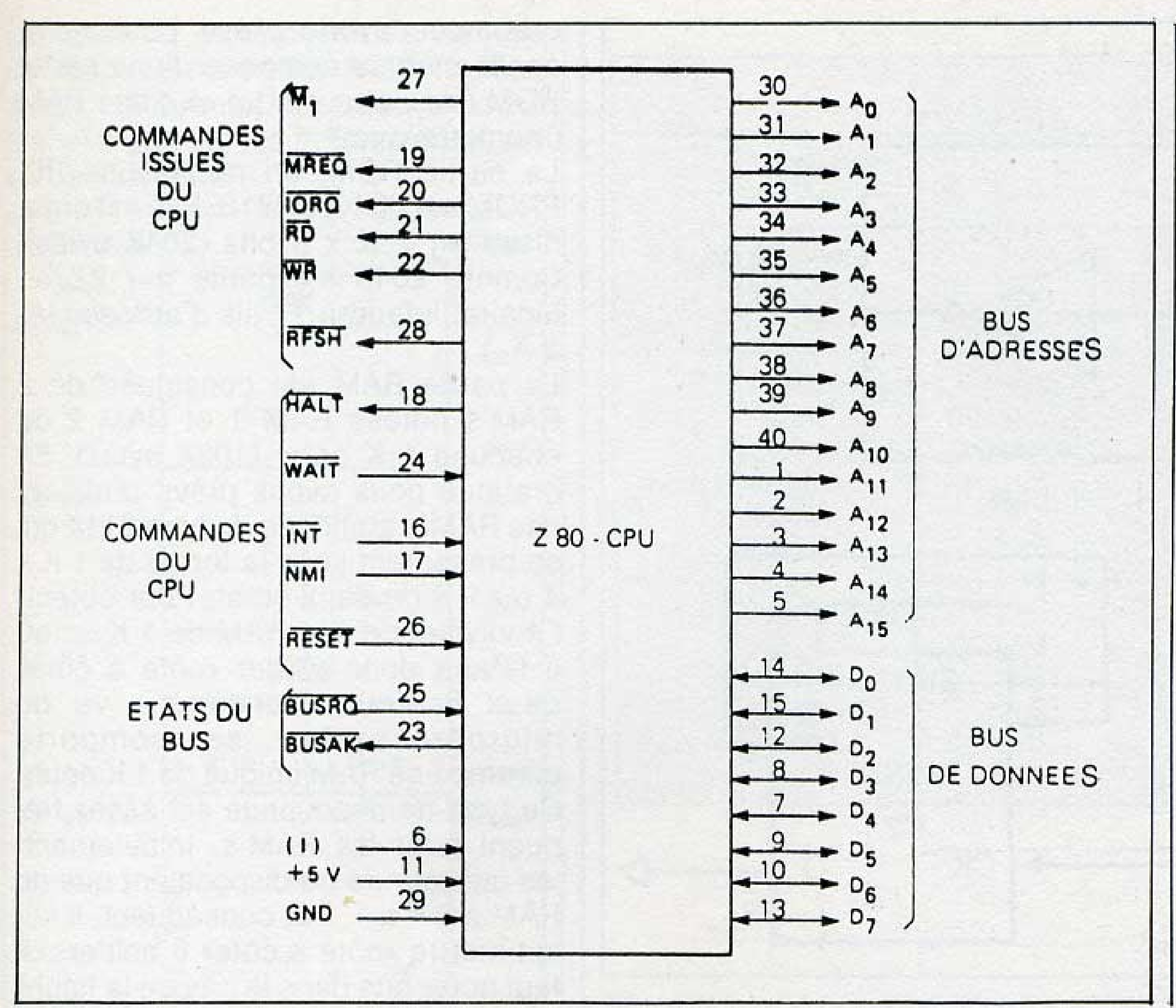


Fig. 44

signaux sont ensuite divisés par 2 avec une bascule. La stabilité exigée pour l'horloge est parfois dictée par le fait que le système doit piloter **d'autres éléments qui requièrent une grande stabilité** de fréquence : lignes de transmission, générateurs de temps, etc...

Un autre avantage, loin d'être négligeable, et qui justifie bien souvent le léger surcroît de dépense est qu'une horloge pilotée par un quartz permet d'utiliser le microprocesseur au «plus près» de sa limitation en fréquence. Puisque, par définition l'horloge est stable, elle peut être calée à la valeur de la limite d'utilisation, sans craindre que la dispersion des composants ou même une éventuelle dérive de l'oscillateur, n'amène **la fréquence en dehors de la plage de fonctionnement** du microprocesseur.

Dans les dispositifs «version économique» comme les calculatrices ou les systèmes peu sophistiqués, le circuit d'horloge peut être réalisé à partir uniquement de réseaux passifs du

trer dans les circuits intégrés digitaux de la famille T.T.L. Cependant nous retrouvons notre classique tension d'alimentation «5 volts».

Il est bon de savoir que la tension d'alimentation unique «5 volts» réalise un très grand progrès dans l'évolution technologique des microprocesseurs. Initialement, un système nécessitait 2 ou 3 tensions (+ 5 V, − 5 V, + 12 V par exemple) ce qui n'était pas sans poser quelques probièmes d'interfaçage avec les autres éléments.

Aujourd'hui, la plupart des microprocesseurs et les composants associés (mémoires, circuits d'interface, etc...) sont alimentés uniquement en 5 volts et dans la majorité des cas les entrées-sorties sont «compatibles T.T.L.».

Dans les conditions normales d'emploi (5 V et 25° ambiante) le courant consommé est de 200 mA (valeur typique) ce qui correspond à une dissipation de 1 watt.

La tension doit être comprise dans une plage de 5 V ± 5 %.

3. L'horloge :

Seule la broche 6 n'a pas encore été cataloguée. C'est l'entrée «horloge» ou «clock».

Un générateur d'impulsions, extérieur au boîtier, fournit un signal carré qui constitue la base de temps de tout système séquentiel. La fréquence maximale à laquelle un microprocesseur peut travailler, constitue pour celui-ci l'une des caractéristiques les plus importantes et permet d'en déduire le temps élémentaire (T.E. est l'inverse de la fréquence).

Ainsi le Z 80 possède 3 versions : (fig 45).

Désignation	F. Max.	Temps élémentaire T.E.
CPU Z 80 normal	2,5 MHz	0,4 µs
CPU Z 80 rapide ou A*	4 MHz	0,25 µs
CPU Z 80 amélioré ou B	6 MHz	0,166 µs

* Le MPF-IB est équipé d'une version Z-80 R-A

Fig. 45

Est-ce qu'une base de temps (ou horloge) très stable est toujours requise dans un système micro-processeur ? Oui, dès que le système est quelque peu évolué. Dans ce cas **le générateur d'horloge est piloté à partir d'un quartz.** De plus pour obtenir une quasi parfaite symétrie des signaux (50 %, 50 %), on utilise un quartz de fréquence double ; les

type R.C ou L.C. Dans ce cas, la fréquence **doit être un peu plus éloignée** de la fréquence limite, et il faut choisir des composants tels que compte tenu des dérives et dispersions, la fréquence reste toujours à l'intérieur de la plage de fonctionnement indiquée par le constructeur (fig 46).

La figure 47 indique le circuit d'horloge utilisé dans le MPF-IB. La fréquence de résonance du quartz est de 3,5795 MHz. Le circuit oscillateur est constitué de deux portes Nand-Trigger 74LS14. Le signal est appliqué sur l'entrée «clock» d'une bascule 74LS74, ce qui permet d'obtenir en sortie Q, un signal de fréquence 1,79 MHz (divisée par 2) avec un rapport cyclique 50/50 parfaitement stable.

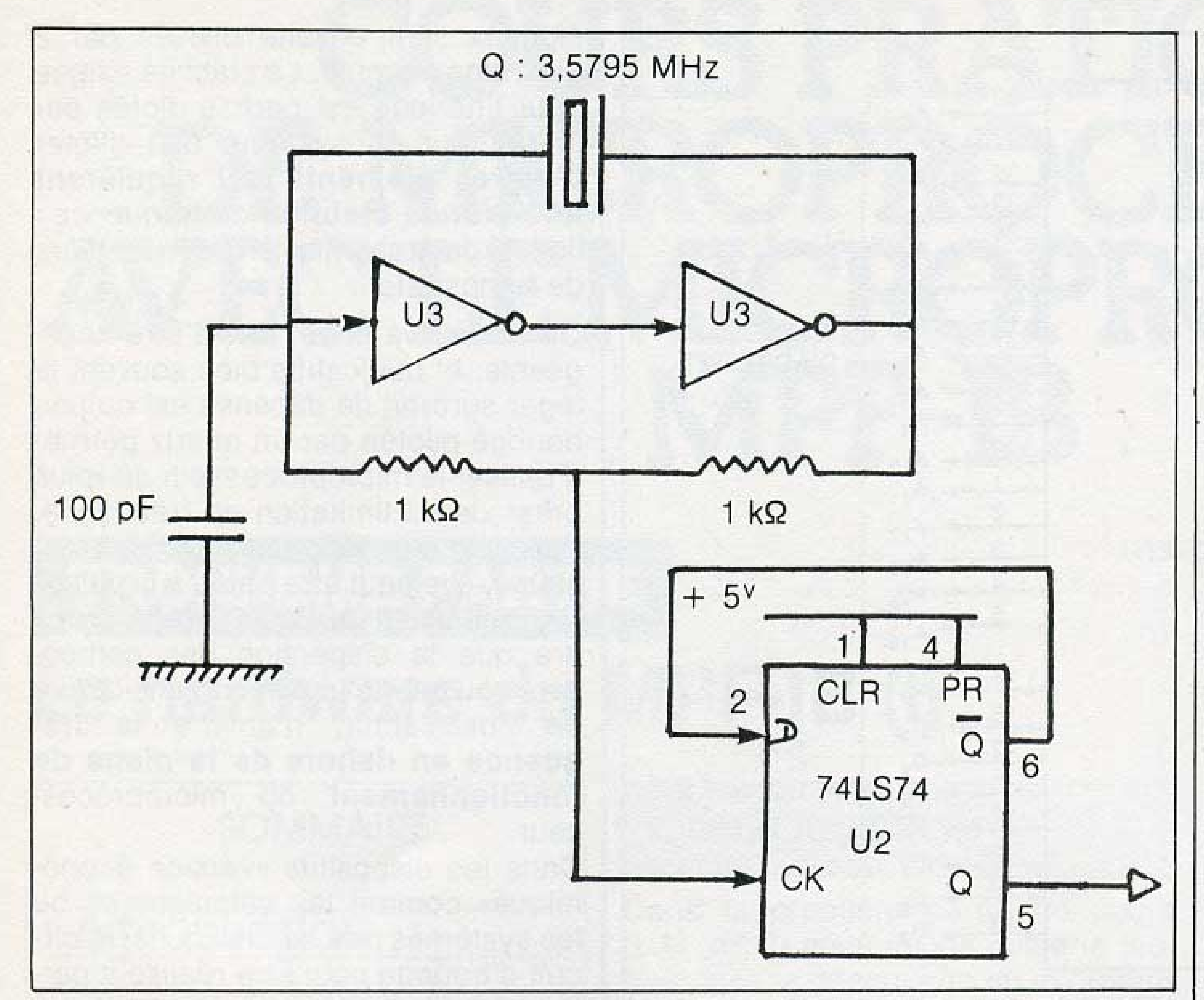


Fig. 47

Le temps élémentaire T.E. est de 0,56 microsecondes.

Quel que soit le type de générateur utilisé, le bon fonctionnement du CPU Z 80 nécessite des flancs d'horloge raides et un niveau haut voisin du « + 5 volts». Cette contrainte (minime) apparaît toujours dans les feuilles de caractéristiques du constructeur qui indique une solution pour y parvenir (fig 48).

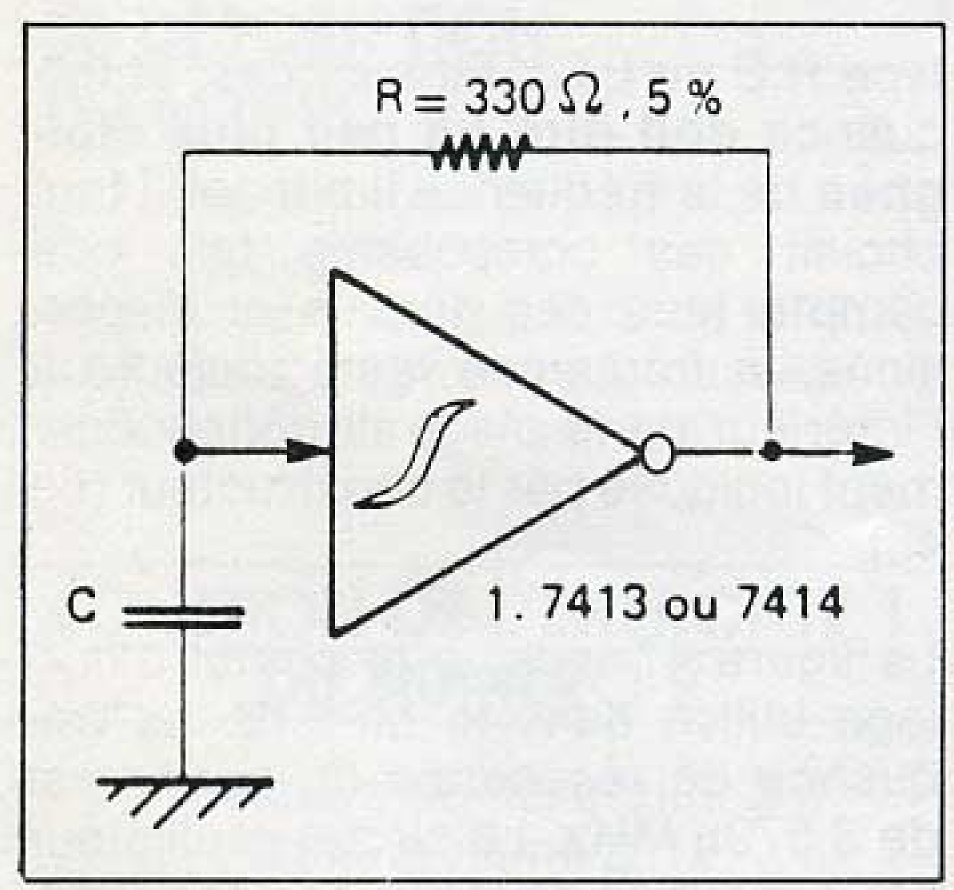


Fig. 46

Le «clock driver» ainsi préconisé (figure 48) est généralement constitué d'une porte T.T.L. (Inverseur ou porte en collecteur ouvert) chargée par une résistance de pull up de 330 Ω (± 5 %) reliée au + 5 volts.

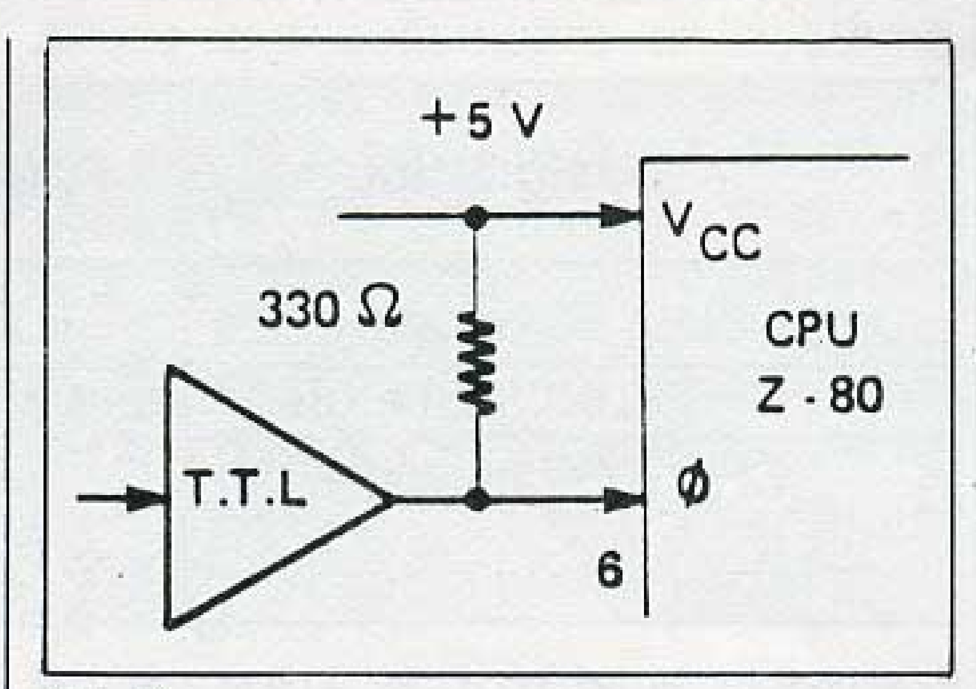


Fig. 48

4. Le bus d'adresses :

La figure 44 indique les 16 sorties réservées à l'adressage. Les sorties sont notées A_0, A_1... A_{15}. Le bit le moins significatif est A_0. Le bit de poids le plus fort est A_{15}. **Chaque sortie peut délivrer un courant de 1,9 mA.** Quand un courant plus important est nécessaire, il faut utiliser un «driver» qui est un amplificateur de courant.

Les 16 lignes d'adresses peuvent adresser une mémoire de 64 K octets. Dans les applications courantes, généralement une capacité mémoire de 2 à 8 K octets est suffisante.

Pour bien comprendre la liaison entre le microprocesseur et la mémoire, nous allons étudier une réalisation pratique de taille moyenne.

L'ensemble mémoire (voir figure 51) que nous avons prévu pour notre application se compose d'une partie ROM (mémoire morte) et d'une RAM (mémoire vive).

La partie ROM, en réalité une REPROM est du type 2716. Elle est organisée en 2 K x 8 bits (2048 bytes). Comme 2048 s'exprime par 2^{11} en binaire, il faudra 11 fils d'adresse (A_0 à A_{10}).

La partie RAM est constituée de 2 RAM's notées RAM 1 et RAM 2 de chacune 1 K octet (1024 bytes). En pratique nous avons prévu d'utiliser des RAM's statiques du type 2114 qui se présentent sous la forme de 1 K x 4 ou 1 K de demi-octet. Pour obtenir l'équivalent d'une RAM de 1 K octet, il faudra donc placer «côte à côte» deux boîtiers : l'ensemble vu du microprocesseur se comporte comme une RAM unique de 1 K octet. Ce type de découpage est assez fréquent pour les RAM's. Initialement, les utilisateurs ne disposaient que de RAM's 1 K x 1. Par conséquent, il fallait mettre «côte à côte» 8 boîtiers. Il faut noter que dans le cas de la figure 51 nous devons considérer que chaque fil d'adresse «voit» deux charges (figure 49).

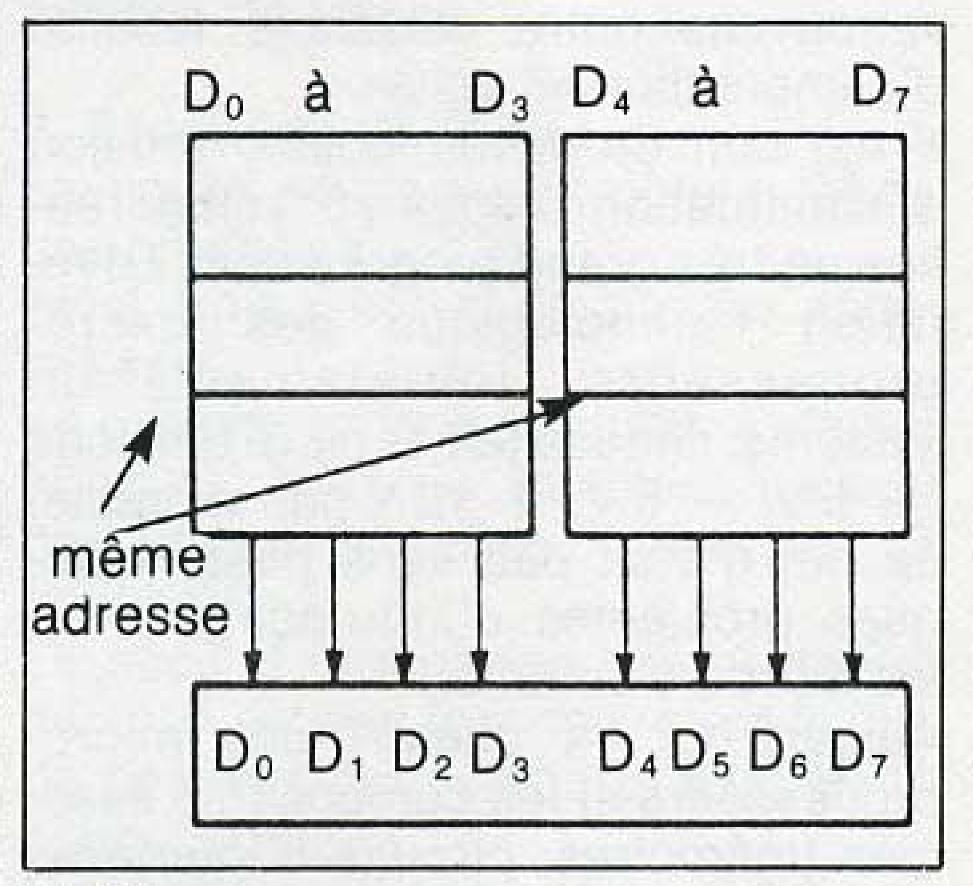


Fig. 49

Nous donnons ci-après le brochage partiel des 2 composants mémoires utilisés (figure 50).

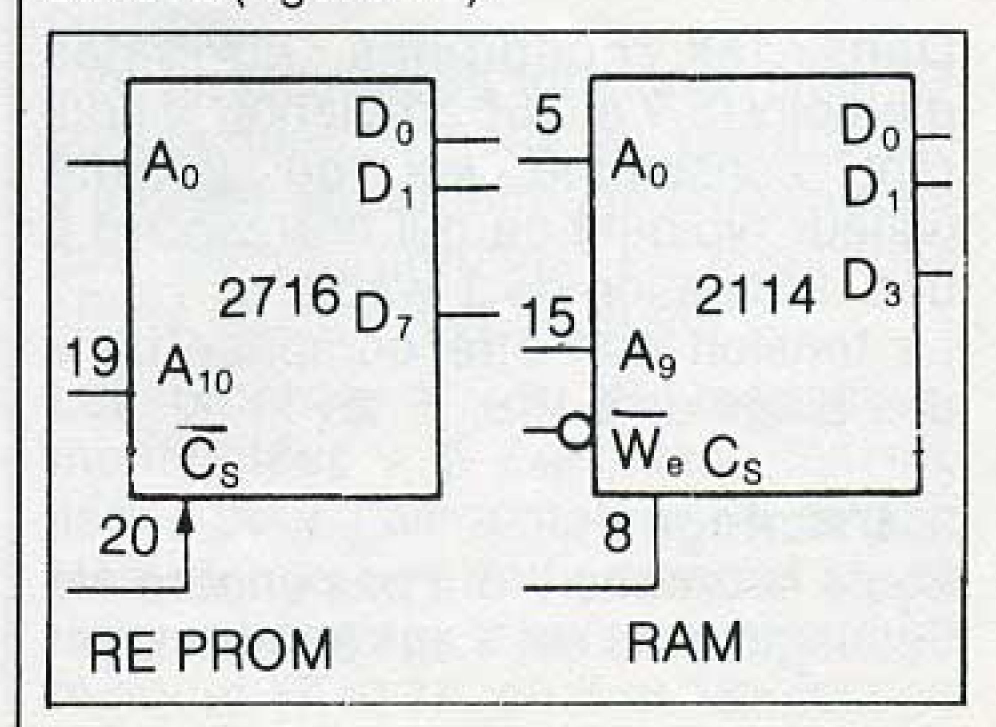


Fig. 50

La figure 51 schématise l'ensemble de l'espace mémoire. A droite nous avons indiqué les adresses en décimal et à gauche les mêmes adresses en hexadécimal.

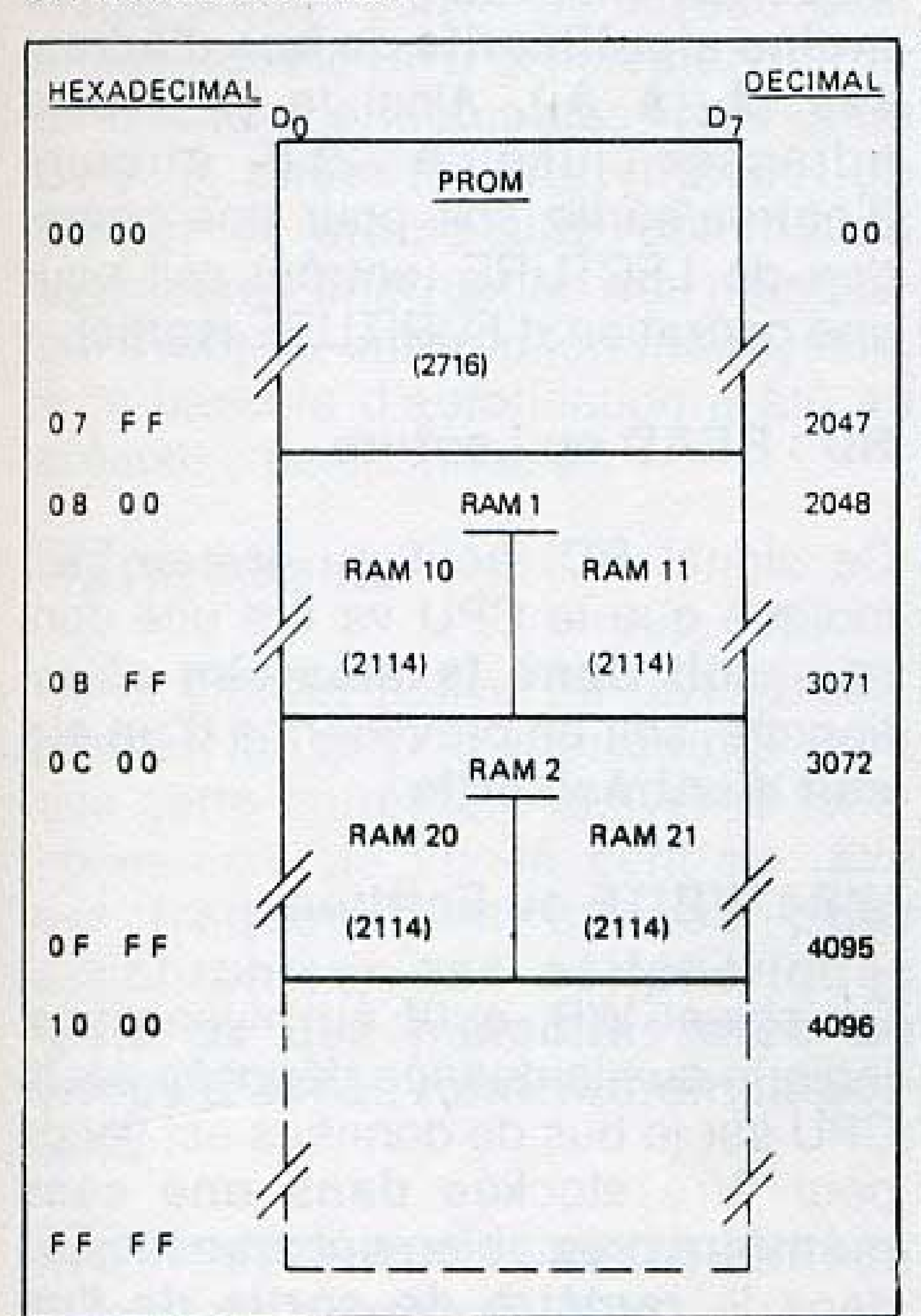


Fig. 51

La première adresse de la PROM est 00 00 en hexadécimal tandis que la dernière est 07 00H.

Quant à la mémoire vive qui est contigüe à la mémoire morte, elle commence à l'adresse 08 00H pour se terminer à l'adresse 0B FF pour la RAM 1 et de 0C 00 à 0F FF pour la RAM 2.

Les codes hexadécimaux ne sont pas très explicites pour résoudre notre problème d'adressage. Nous lui préfèrerons, une fois n'est pas coutume, le code binaire. C'est ainsi que nous avons établi le tableau de la figure 52.

La première remarque que nous pouvons faire est que les quatre lignes d'adresse de poids forts (A_{15}, A_{14}, A_{13} et A_{12}) ne sont pas utilisées pour l'adressage de la mémoire. Ceci était prévisible puisque nous n'adressons que 4 K octets, soit 4096 bytes ce qui se représente par 2^{12} donc ne nécessite que 12 lignes.

En «ignorant» les lignes égales ou supérieures à A_{12}, nous constatons que la PROM sera sélectionnée quand A_{11} sera à «0», la RAM 1 (RAM's 10 et 11) quand A_{11} ET A_{10} sont respectivement à «1» et «0» et la RAM 2 (RAM's 20 et 21) quand A_{11} ET A_{10} sont tous deux à «1».

Examinons bien le brochage donné

		A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7 à A_0
PROM	1ère adresse 00 00	0	0	0	0	0	0	0	0	0
	dernière adresse 07 FF	0	0	0	0	0	1	1	1	1
RAM 1	1ère adresse 08 00	0	0	0	0	1	0	0	0	0
	dernière adresse 0B FF	0	0	0	0	1	0	1	1	1
RAM 2	1ère adresse 0C 00	0	0	0	0	1	1	0	0	0
	dernière adresse 0F FF	0	0	0	0	1	1	1	1	1
		← 4 →				← 3 →			← 2 et 1 →	
		← 2ème byte →							← 1er byte →	

Fig. 52

par la figure 50. Nous voyons sur chacun des composants une entrée notée C_s (ce qui signifie «Chip Select» ou Sélection du boîtier). Cette entrée est donc utilisée pour sélectionner un boîtier, c'est-à-dire **une plage donnée dans l'ensemble du plan mémoire.**

Nous avons présenté avec des circuits logiques «simples» la manière de sélectionner un boîtier donné dans le champ mémoire, en n'utilisant que des portes.

Dans le cas du MPF-IB, on utilise des circuits intégrés 74LS139 (2). D'autre part, pour permettre d'adresser les

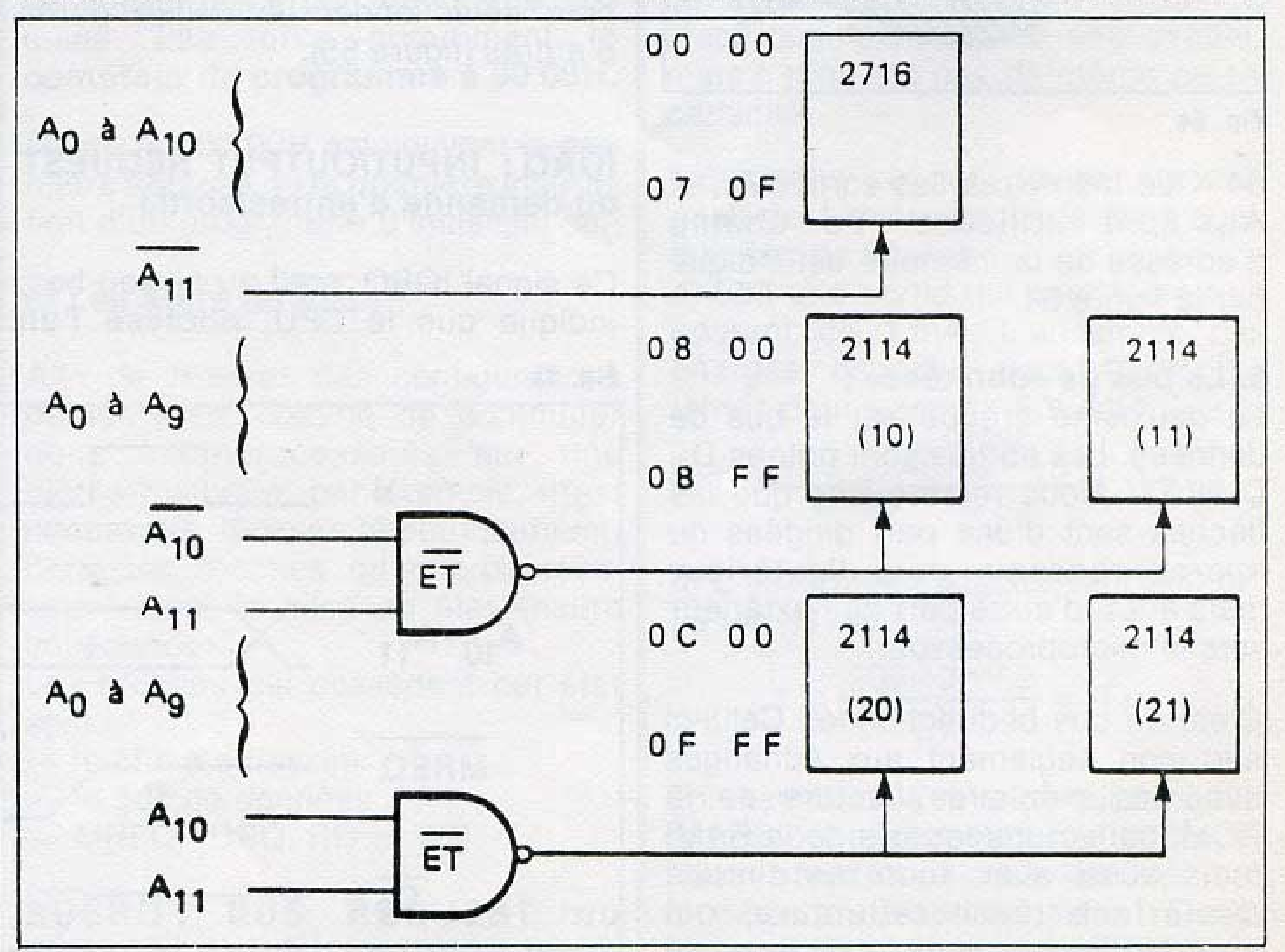


Fig. 53

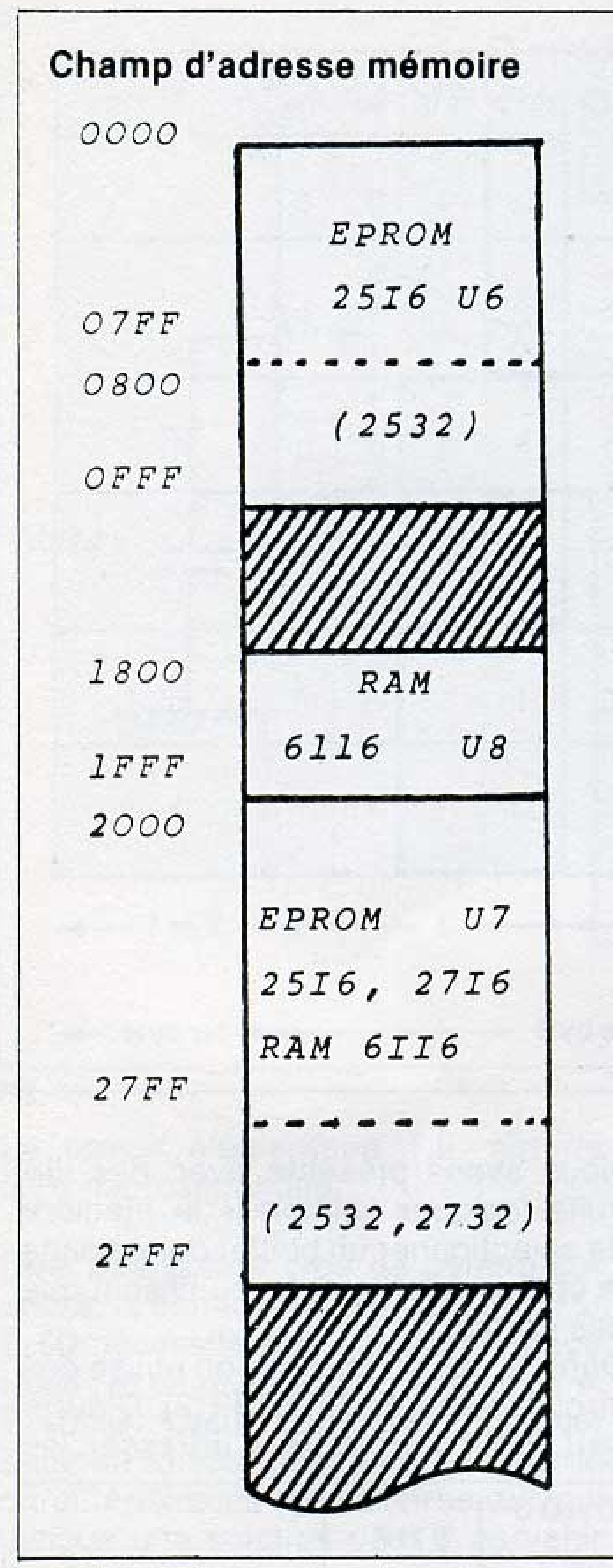


Fig. 54

64 K de mémoires, les sorties A_{14} et A_{15} sont utilisées. Le champ d'adresse de la mémoire est indiqué par la figure 54.

5. Le bus de «données» :

Le deuxième groupe est le bus de données. Les sorties sont notées D_0, D_1,... D_7. Nous remarquons que les flèches sont d'une part dirigées du microprocesseur vers l'extérieur mais aussi d'autre part de l'extérieur vers le microprocesseur.

C'est un bus bi-directionnel. Celui-ci sert non seulement aux échanges avec les mémoires (lecture de la ROM, ou lecture/écriture de la RAM) mais aussi avec tous les circuits d'interface (écriture/lecture) qui assurent la communication avec le «monde» extérieur.

6. Le bus de commande :

Le «bus de commande» qui regroupe 13 commandes peut se décomposer en trois sous-groupes :

— un sous-groupe de 6 sorties qui concerne les signaux de commande issus de l'unité de contrôle.

— un sous-groupe de 5 (4 entrées, 1 sortie) qui regroupe des commandes qui agissent sur le CPU.

— un sous-groupe de 2 broches relatif aux états du bus.

a) Commandes issues de l'unité de contrôle :

$\overline{M_1}$: Premier cycle machine

Ce signal M_1, actif au niveau bas, indique le premier cycle dans le déroulement d'une instruction qui peut en comporter au minimum 1 et au maximum 6. Le premier cycle correspond toujours **à la recherche du code opération** de l'instruction à exécuter.

$\overline{\text{MREQ}}$: Memory Request ou Demande d'accès à la mémoire

Ce signal MREQ, actif au niveau bas, indique que l'adresse déposée sur le bus d'adresse est valide et qu'une opération de lecture ou d'écriture peut avoir lieu. Ce signal (ou son complément) est souvent utilisé comme l'une des entrées d'une porte pour sélectionner un boîtier parmi d'autres (figure 55).

$\overline{\text{IORQ}}$: INPUT/OUTPUT REQUEST ou demande d'entrée/sortie

Ce signal IORQ, actif au niveau bas, indique que le CPU, **adresse l'un des périphériques** (à ne pas confondre avec une adresse mémoire). Pour des raisons d'économie de broches, l'adresse des périphériques est constituée au maximum des 8 bits **les moins significatifs du bus d'adresses (A_0 à A_7).** Ainsi le CPU peut adresser jusqu'à 256 circuits d'entrée/sortie soit pour une opération de LECTURE (entrée) soit pour une opération d'ECRITURE (sortie).

$\overline{\text{RD}}$: READ ou Lecture

Ce signal RD, actif au niveau bas, indique que le CPU va lire une donnée, **soit dans la mémoire** sélectionnée, soit en provenance **d'un circuit d'entrée/sortie.**

$\overline{\text{WR}}$: WRITE ou Ecriture

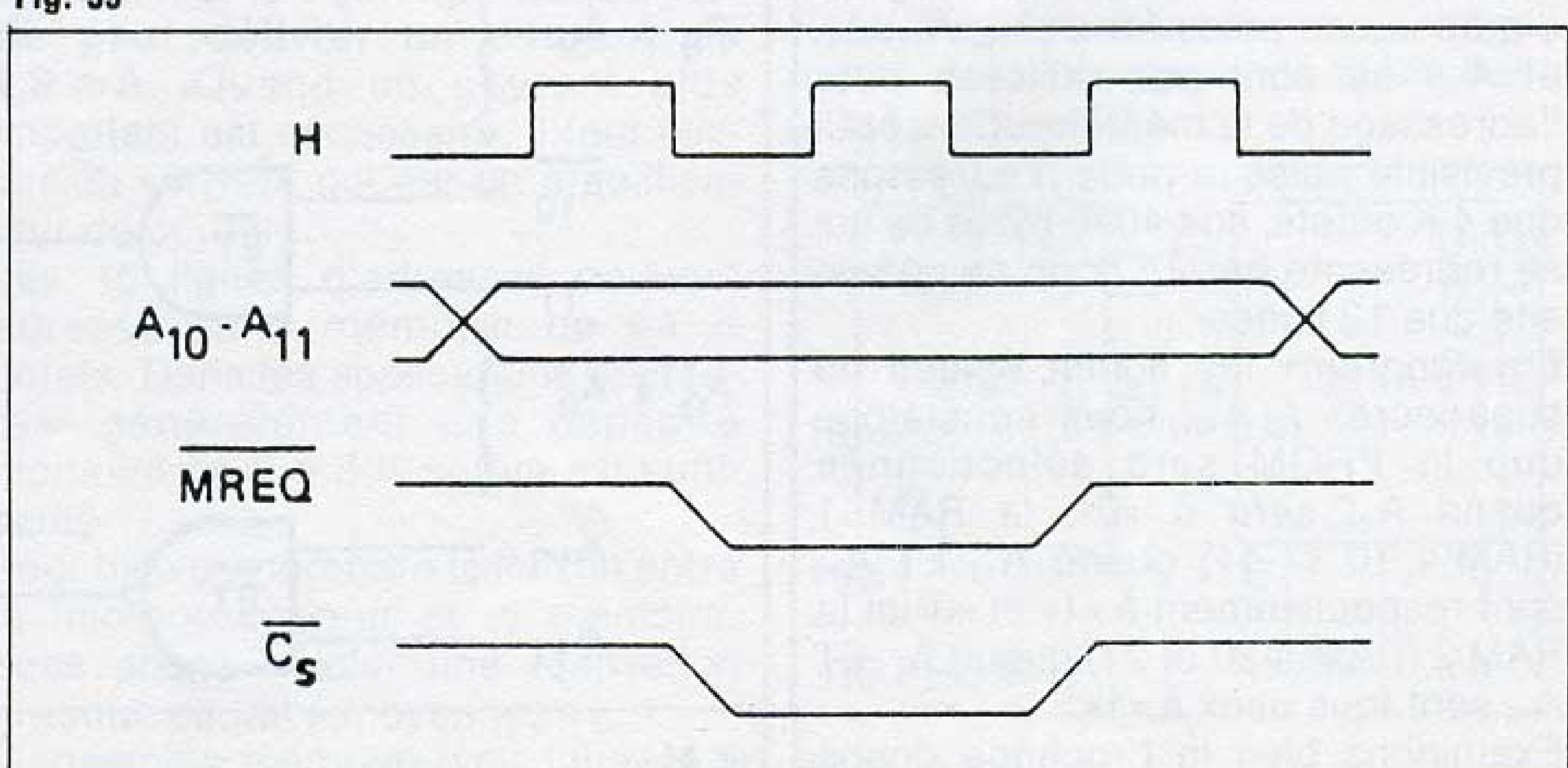
Ce signal WR, actif au niveau bas, indique que la donnée déposée par le CPU sur le bus de données est valide pour être stockée **dans une case mémoire** préalablement adressée ou dans le **registre de sortie de l'un des circuits d'entrée/sortie.**

$\overline{\text{RFSH}}$: REFRESH (ou rafraîchissement)

Ce signal RFSH, actif au niveau bas, est utilisé pour le rafraîchissement de toutes les mémoires dynamiques. Rappelons que le rafraîchissement des mémoires (dynamiques uniquement) consiste **à lire le contenu d'une ou plusieurs cases mémoires et de le réécrire dans le même emplacement.** Cette opération d'écriture/lecture a pour but de compenser les pertes de charge de ces cases mémoires constituées par un «condensateur».



Fig. 55



b) Les commandes du CPU :

$\overline{\text{HALT}}$: Etat d'arrêt

Cette sortie, active au niveau bas, indique que le CPU est en état d'arrêt (en réalité il effectue des «NOP»). Le CPU a exécuté une instruction d'arrêt et attend une commande extérieure : interruption non masquable (NMI) ou interruption masquable, si la bascule d'autorisation a été au préalable positionnée en 1.

$\overline{\text{WAIT}}$: Attente

Cette entrée, active au niveau bas, permet au CPU d'interrompre (tant que cette entrée est active) le fonctionnement de l'unité centrale. Elle est fréquemment utilisée **pour synchroniser des périphériques lents ou des mémoires avec un temps d'accès relativement important.**

$\overline{\text{INT}}$: INTERRUPT REQUEST ou Demande d'interruption

Cette entrée, active au niveau bas, provient en général d'une unité périphérique pour **demander un détournement du déroulement normal** du programme en cours.
L'autorisation d'accepter cette demande s'effectue sous le contrôle du programme (donc du programmeur).

Lorsque la **demande est acceptable et acceptée,** le CPU **exécute intégralement l'instruction en cours, sauvegarde** le programme principal et **«saute» au sous-programme** relatif à l'interruption. Quand la sous-routine est exécutée, le déroulement du programme principal reprend là où il avait été interrompu.

$\overline{\text{NMI}}$: NON MASQUABLE INTERRUPT (Interruption non masquable)

Cette entrée, active au niveau bas, indique au CPU qu'il **doit impérativement effectuer un détournement** du programme en cours vers une sous-routine. Cette demande est dite de **plus forte priorité** et souvent utilisée pour sauvegarder l'état d'un équipement.
Etudions un exemple.
Une caisse enregistreuse est utilisée en sortie d'un supermarché pour comptabiliser le montant des achats «clients». Si une coupure de secteur intervient, ne fusse que quelques secondes... et sans précaution, l'ensemble de la transaction disparaît... et tout est à reprendre au retour du secteur.
Pour pallier à cet inconvénient, la caisse possède une mémoire secourue (mémoire RAM alimentée par une batterie). Un circuit «surveille» la tension secteur, et quand celle-ci descend au-dessous d'un seuil critique, l'entrée N.M.I. est activée. Aussitôt, le CPU suspend son programme, «saute» au programme de sauvegarde : les éléments essentiels (totaux, états des registres, etc...) sont transférés dans la mémoire secourue. Au retour du secteur, l'opération inverse se produit et... tout se déroule normalement.
Il faut noter que les condensateurs d'alimentation doivent avoir une capacité suffisante pour assurer un fonctionnement normal du système pendant le temps d'exécution du programme de sauvegarde (de l'ordre de quelques dizaines de millisecondes).

$\overline{\text{RESET}}$: Remise à zéro ou initialisation

Cette entrée, active au niveau bas, a pour but d'initialiser le CPU et de désactiver un certain nombre de bascules. Elle force notamment **le compteur de programme à 00 00H.**

De ce fait 00 00H est souvent la première adresse ou la première instruction d'un programme d'initialisation.

c) Les états du bus :

Afin de réaliser des configurations particulières, comme de commuter deux microprocesseurs sur une même mémoire, par exemple, il est nécessaire d'isoler (électriquement) certaines broches du microprocesseur : c'est la mise en état «haute impédance».
Les broches qui possèdent cet état sont :
— le bus d'adresses
— le bus de données
— MREQ, IORQ, RD et WR

$\overline{\text{BUSRQ}}$: BUS REQUEST ou demande de mise en «haute impédance»

Cette entrée, active au niveau bas, indique au CPU une demande de mise en «haute impédance». Celle-ci ne pouvant avoir lieu immédiatement, elle ne sera honorée que lorsque **le cycle en cours sera terminé.**

$\overline{\text{BUSAK}}$: BUS ACKNOWLEDGE (acceptation de la demande à haute impédance)

Cette sortie, active au niveau bas, indique que les broches «3 états» sont effectivement isolées du reste du CPU et que le circuit périphérique demandeur peut utiliser les bus d'adresses, de données et de commandes.

7. Quelques circuits annexes :

a) Circuits «BUFFERS»

Lorsque le nombre de circuits à commander augmente, le courant nécessaire pour les commander augmente, et il arrive à un certain moment que le microprocesseur ne peut plus fournir le courant demandé (celui-ci est limité à 1,9 mA).

En plus des boîtiers «mémoires» et «circuits de décodage», il existe d'autres charges qu'il ne faut pas oublier : les liaisons.
Une liaison de l'ordre de quelques centimètres présente une résistance électrique faible, quasi négligeable, mais il n'en est pas de même de sa capacité.

Examinons ce que devient un créneau de 1 micro-seconde dans quelque cas de figures :
a) Soit une sortie qui peut fournir un courant de 1 mA. L'ensemble des charges capacitif vaut 40 pF. Le temps pour attendre 5 V est :

$$t = \frac{C.V}{I} = \frac{40.10^{-12}.5}{10^{-13}} = 0{,}2\ \mu s$$

b) L'ensemble capacitif vaut 200 pF. Le temps est :

$$t = \frac{200.10^{-12}.5}{10^{-3}} = 1\ \mu s$$

En examinant la relation qui donne le temps nécessaire pour attendre V, nous constatons que celui-ci est inversement proportionnel à I. Donc, **lorsque la charge capacitive augmente, des circuits de puissances**

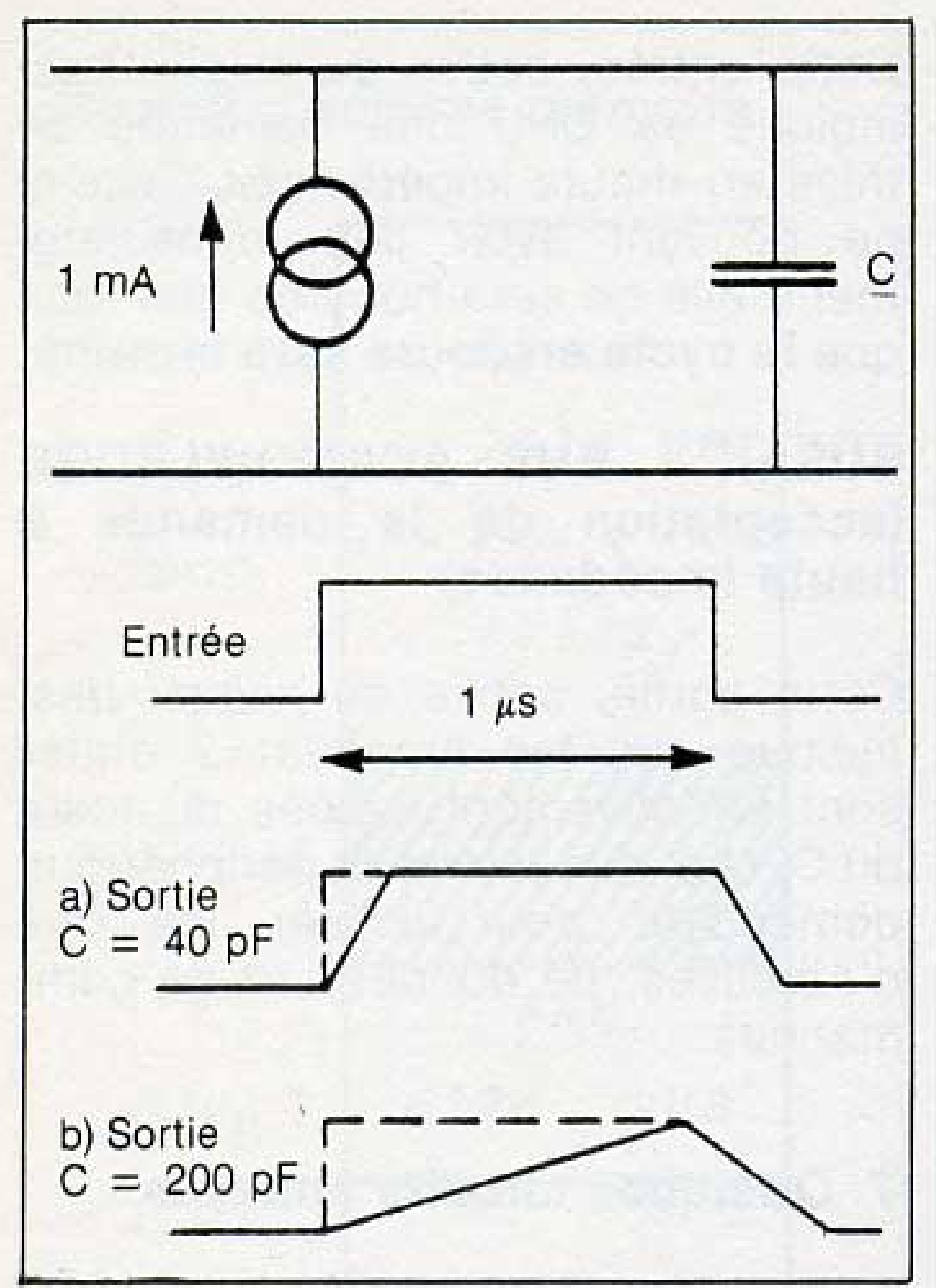


Fig. 56

capables de fournir un courant important seront nécessaires.
De tels circuits, qui n'ont qu'un rôle d'amplificateur de courant sont désignés sous le nom de «BUFFER» ou «LINE DRIVER». Ainsi le circuit 74LS240 est capable de fournir un courant de 24 mA par sortie (il en possède 8) et 40 mA en version 74S240.
Habituellement, ces circuits sont de type «TRI-STATE». C'est-à-dire que sous l'action d'une commande appropriée «OE», les sorties deviennent «Haute Impédance», c'est-à-dire que tout se passe comme si le circuit était enlevé (figure 57).

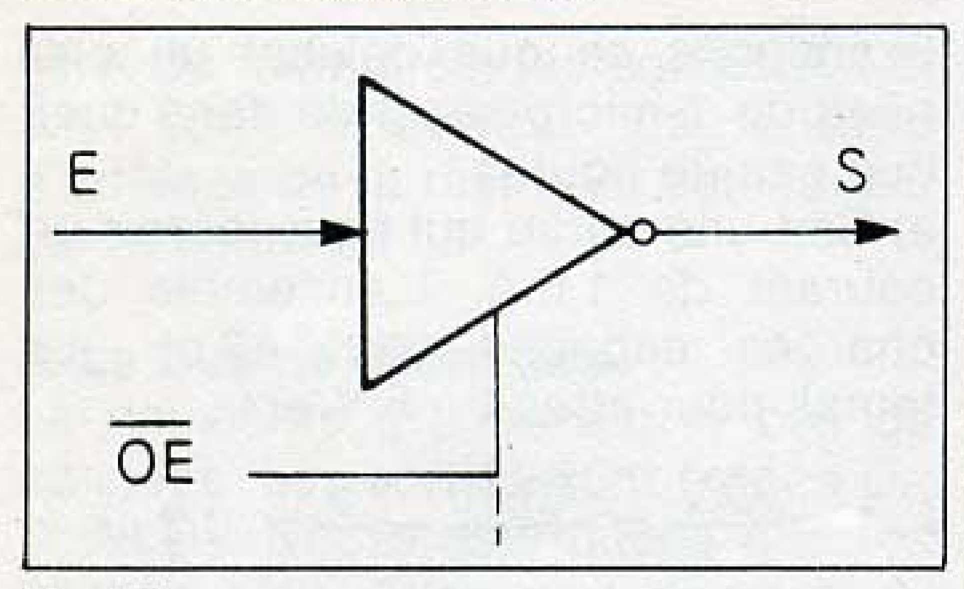


Fig. 57

Les circuits appropriés dans le cas du bus d'adresses sont par exemple tels que le 74S240, 74S241 et 74S244 ou le 74LS240, 74LS241 et 74LS244.
Ces circuits sont unidirectionnels donc ne peuvent être utilisés pour le **bus de données qui nécessite un circuit «bi-directionnel».**
La figure 59 représente deux cellules, et la logique associée.
Chaque cellule comporte deux amplificateurs. L'un dans le sens A vers B, l'autre de B vers A. Chacun d'eux reçoit une commande C_1 ou C_2 qui détermine le fonctionnement selon la table de vérité de la figure 58.
Le circuit représenté est le 74LS245 qui comporte 8 buffers bidirectionnels avec sur les deux voies de communication, la possibilité d'être commutée en haute impédance.
La figure 60 représente un schéma d'ensemble de la mise en œuvre du microprocesseur Z 80A (Extrait de LED n°3 de Ph. Faugeras).

b) Circuit «RAM Secourue» :
Dans ce type d'application, la RAM utilisée est du type C.MOS à très faible consommation. Elle présente deux types de fonctionnement suivant la tension d'alimentation appliquée.

Entrées		Sortie
$\overline{E}$	D_R	S
0	0	Sens B → A
0	1	Sens A → B
1	X	Haute impédance

Fig. 58

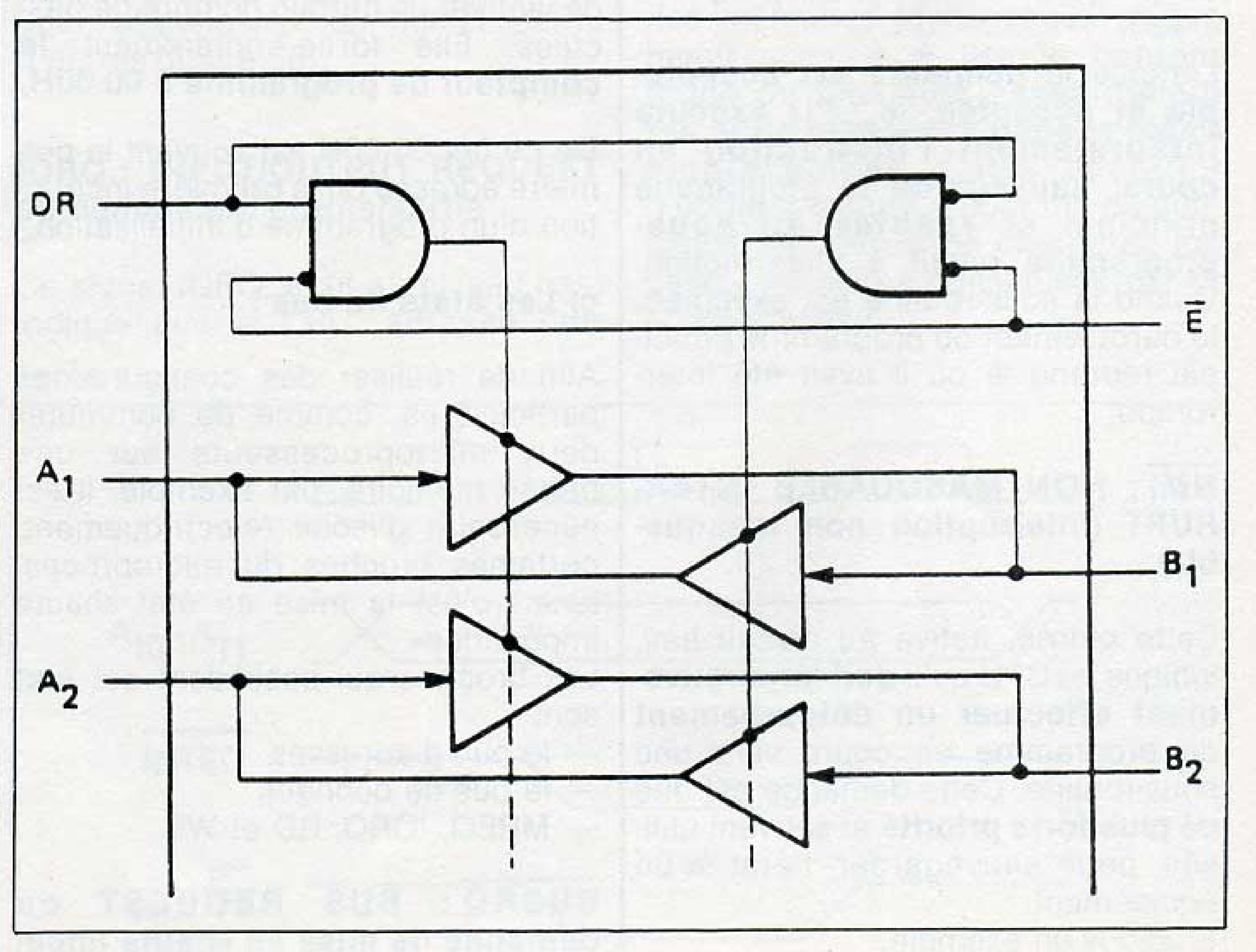


Fig. 59

Quand la tension est de 5 volts, la RAM CMOS se comporte comme une mémoire à accès aléatoire normale. En absence de tension, son contenu disparaît. Sous 5 volts, le courant est de l'ordre de 7 à 10 mA (RAM 6514 de Harris ou Intersil).
Quand la tension «tombe» de 5 volts à 2 ou 3 volts, la mémoire se trouve en «hibernation» ou en «Stand by». Son contenu ne s'efface pas, mais elle ne peut être adressée ni pour une lecture ni pour une écriture. Elle conserve l'intégralité de son contenu qui redeviendra accessible et disponible quand la tension reviendra à son «niveau normal».
Le courant d'alimentation, ou «courant de rétention» pour la maintenir ainsi en «veilleuse» n'est plus que de l'ordre de 5 à 10 µA : c'est-à-dire plus de mille fois moins qu'en régime normal. Une petite batterie rechargeable de faible capacité peut ainsi sauvegarder le contenu de la mémoire pendant plusieurs mois.
Le schéma de la figure 61 donne un exemple de réalisation. Les RAM's CMOS utilisées sont du type 6514 qui présentent l'avantage d'être compatibles, interchangeables avec les 2114.
La commutation entre la tension normale + 5 V et la tension de rétention 2,4 V s'effectue automatiquement grâce aux diodes D_1 et D_2.

Fig. 60

En fonctionnement «normal» le courant I_N alimente les RAM's, ce courant est fourni par l'alimentation principale au travers de D_1.

Notons que nous avons dû élever le 5 V à 5,6 V pour tenir compte de la chute de tension dans D_1. La résistance R_1 de 1 kΩ assure la recharge de la batterie avec un courant de 2 mA environ.

Lorsque E disparaît, la batterie B de 2,4 volts alimente au travers de R_1 les RAM's. La présence de D_1 empêche la batterie de débiter dans le reste des circuits.

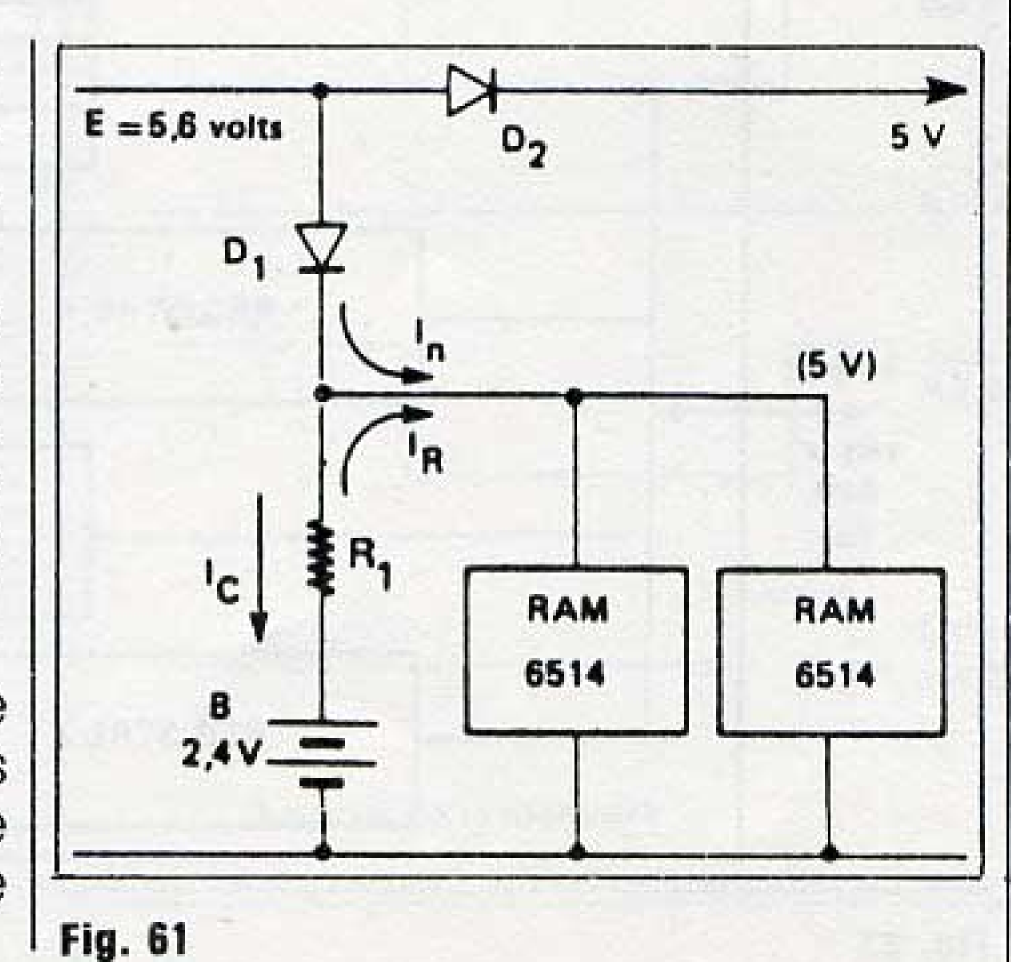


Fig. 61

III. DESCRIPTION DU MPF-IB

III.1 Présentation physique

La figure 62 (page suivante) montre le schéma d'implantation des composants qui constituent le MPF-IB. En haut à gauche, le Z 80 (U_1) avec son connecteur de sortie P1. Sur ce dernier, tous les signaux, tels que nous venons de les décrire, sont disponibles pour des extensions. Celles-ci peuvent être soit un module complémentaire comme l'imprimante thermique PRT-MPF, un programmateur d'EPROM's ou tout autre circuit d'interface.

Les emplacements U6, U7 et U8 sont réservés aux mémoires ROM et RAM, dont le champ d'adresses est donné par la figure 54.

Les supports notés U10 et U11 respectivement prévus pour des circuits d'interface PIO-Z 80 et CTC-Z 80 sont livrés en option. Ceux-ci feront l'objet d'une étude ultérieurement.

Le microprocesseur Z 80^R pour pouvoir fonctionner doit être relié avec des périphériques immédiats : les afficheurs (U16 à U21) ainsi que le clavier. Le 8255 d'INTEL sert de circuit d'interface pour l'ensemble Affichage et Clavier ainsi que pour la liaison série avec le magnétocassette.

III.2 Le 8255 ou P.P.I.

Dans les échanges entre le microprocesseur et la mémoire (ROM ou RAM) la communication peut être directe, tout au plus au travers d'un circuit «buffer» quand le nombre de circuits dépasse la charge acceptable.

Quand il s'agit de communiquer avec d'autres périphériques, comme le clavier, la visu ou tout autre dispositif, le fonctionnement devient plus délicat.

En effet, nous ne disposons que d'un seul canal, le bus de données, lequel est déjà utilisé pour communiquer avec la mémoire.

Dans le cas d'une opération de LECTURE ou d'ECRITURE dans la mémoire, l'opération est rapide : elle s'effectue au cours d'une seule instruction (RAPPEL fig 9, LM 9 p. 61).

Par contre, avec un périhérique, il en va autrement.

Dans le cas d'une information de sortie, il faudra la maintenir pendant un certain temps : exemple l'affichage ou de la commande d'un moteur. De même pour «saisir» une donnée, en provenance du clavier, il n'est pas

Fig. 62

pensable de laisser le microprocesseur à l'«écoute du clavier», en attendant une hypothétique information : **car bien entendu quand il attend, il ne peut rien faire d'autre.**

La fonction essentielle d'un circuit d'Entrée-Sortie est de «stocker temporairement» les informations soit en provenance du CPU vers l'extérieur (affichage, commande du moteur par exemple) soit **de l'extérieur vers le CPU** (clavier, par exemple). Grâce à l'emploi de ce circuit, les problèmes de vitesse et de synchronisation sont quasiment résolus. Le microprocesseur considère le circuit d'entrée-sortie comme une ou plusieurs adresses mémoires dans lesquelles il vient chercher une information ou déposer une donnée.

La figure 63 donne le synoptique simplifié du 8255.
Le circuit 8255 est comme un petit microprocesseur en lui-même.

Le «8255» est constitué de 4 registres identiques à ceux du CPU. Chaque registre peut contenir un byte (8 bits).

Trois des registres (0, 1 et 2) sont connectés à trois groupes de 8 lignes d'entrée ou de sortie. Le quatrième registre (3), adressé comme les trois autres, contient le «mot de contrôle» : c'est ce mot qui détermine la configuration de chaque registre 0 à 2.

Comme l'indique la figure 64, le 8255 contient trois registres qui communiquent avec le monde extérieur dont la répartition est la suivante :
— Registre d'adresse «0» ou «00» PORT A : 8 bits
— Registre d'adresse «1» ou «01» PORT B : 8 bits
— Registre d'adresse «2» ou «02» PORT C inf. : 4 bits (C_0 à C_3), PORT C sup. : 4 bits (C_4 à C_7)

En mode 0, le seul que nous étudierons, chacun des ports, selon l'état du mot de contrôle (voir tableau des configurations en mode 0) est programmé soit en «Entrée» soit en «Sortie». Comme il existe 4 ports (A, B, C_I et C_S) il en résulte 16 combinaisons possibles.

Tous les échanges entre l'un des registres du circuit 8255 et le micro-

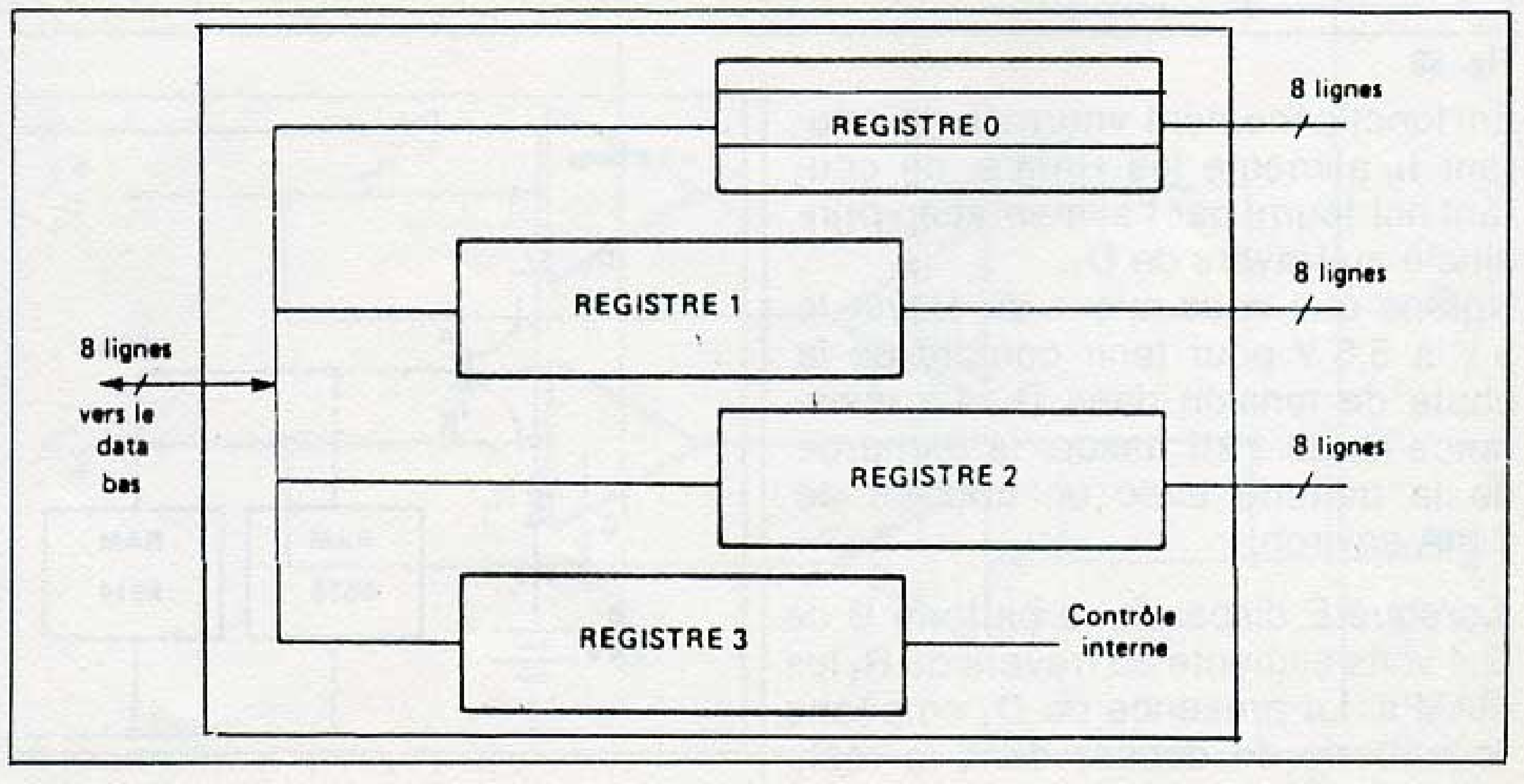


Fig. 63

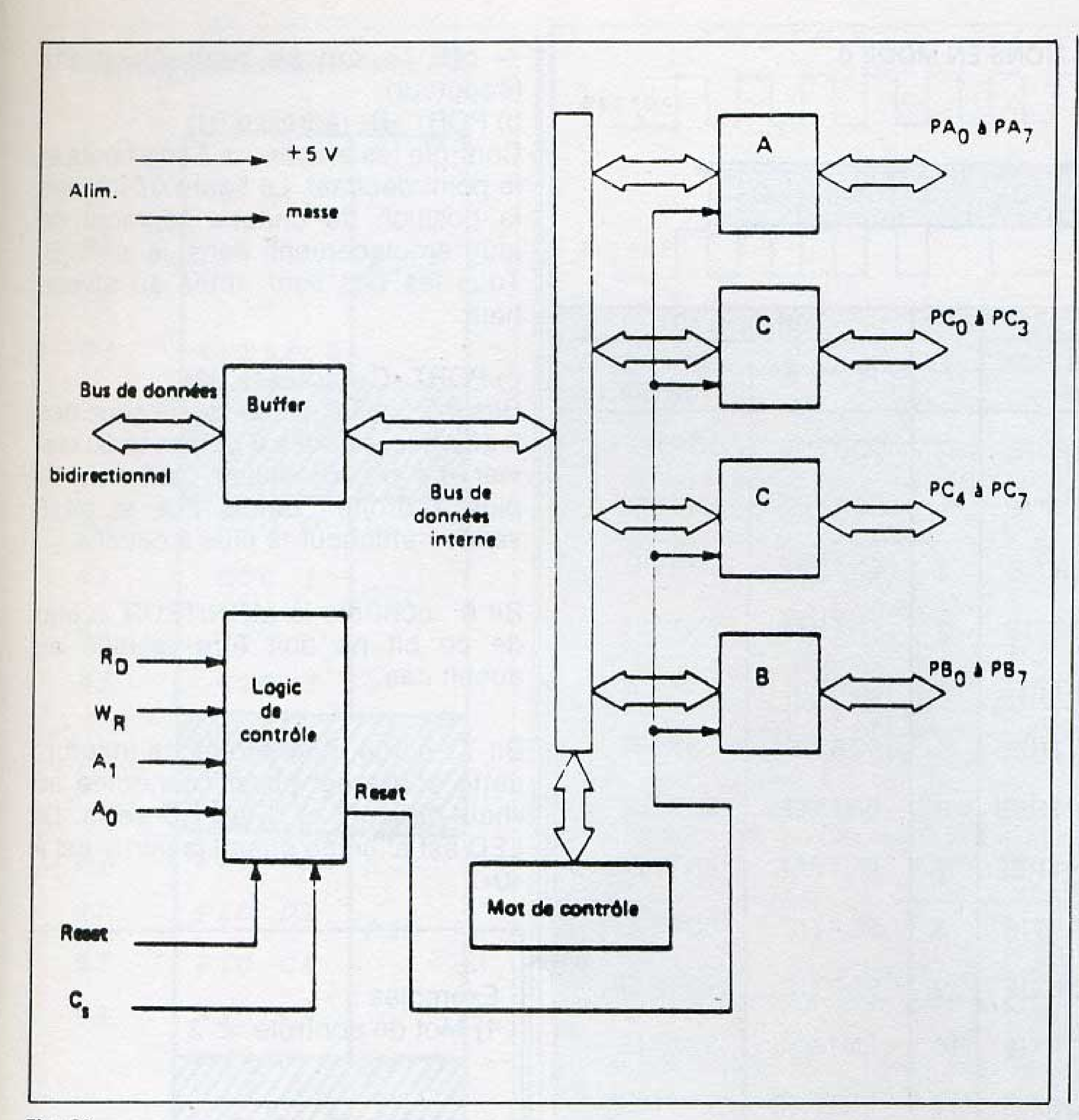


Fig. 64

processeur s'effectuent toujours au travers **de l'Accumulateur A et du bus de Données.**

La sélection du registre du circuit d'interface s'effectue à l'aide des entrées A_0 et A_1, lesquelles sont reliées au bus d'adresses. Ainsi l'exécution de l'instruction «IN A, (2)» consiste à lire le contenu du registre 2 (Ports C), et à transmettre son contenu dans le registre Accumulateur.

D'une manière analogue, l'exécution de l'instruction OUT (3), A consiste à transférer le contenu du registre A dans le registre réservé au mot de contrôle.

Nous venons d'écrire que la sélection d'un des registres de sortie du circuit périphérique se sélectionnait au moyen du bus d'adresses. Dans ce cas comment éviter le conflit avec la case mémoire d'adresses identique ?

Pour transférer le contenu de A dans la case mémoire 03, l'instruction est :

Load 03 ← A ou Ld 03, A

tandis que le transfert du contenu de A dans le registre de sortie 03, l'instruction est :

OUT 03 ← A ou OUT 03,A

C'est le CPU lui-même qui, au moyen du bus de commande, effectue la différentiation, en générant le signal MREQ dans le premier cas (lecture de la mémoire) et le signal IORQ dans le second cas. La figure 65 donne les chronogrammes dans chaque cas.

Dans le cas, comme le nôtre, la sortie IORQ est connectée directement à l'entrée C_S (Chip Select). C'est-à-dire qu'un niveau bas, **sélectionne automatiquement le périphérique.**

Quand plusieurs périphériques sont utilisés, il faut utiliser une logique combinatoire de sélection, dans laquelle entrent les lignes d'adresse et le signal IORQ.

La figure 66 indique deux exemples de configuration avec le mot de contrôle correspondant (voir page suivante).

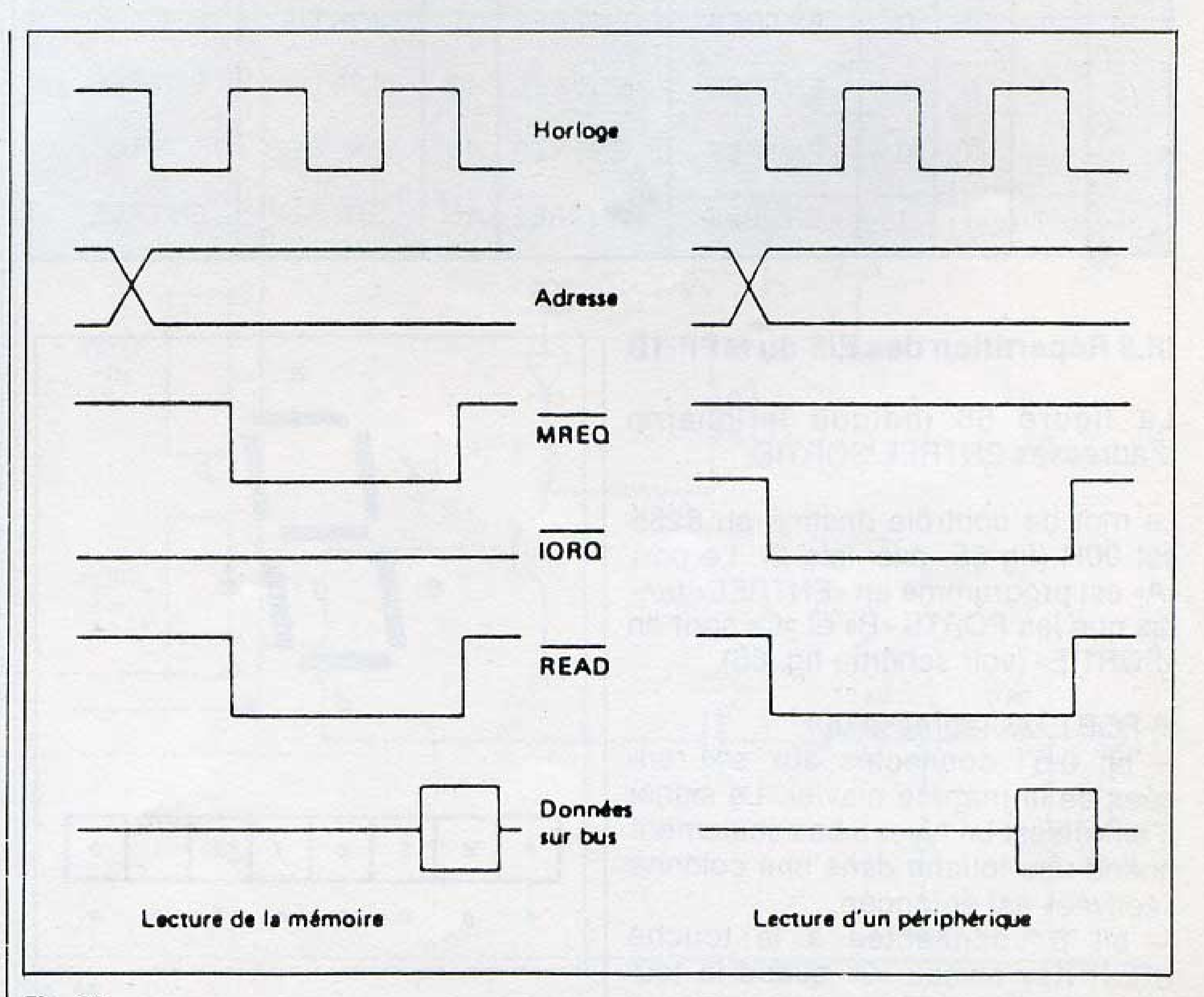


Fig. 65

TABLEAU DES CONFIGURATIONS EN MODE 0

Le «mot de contrôle» est tel que :

1	0	0	D_4	D_3	0	D_1	D_0

avec D_4, D_3, D_1 et D_0 :

A		B		GROUPE A			GROUPE B	
D_4	D_3	D_1	D_0	PORT A	PORT C supérieur		PORT B	PORT C inférieur
0	0	0	0	SORTIE	SORTIE	0	SORTIE	SORTIE
0	0	0	1	SORTIE	SORTIE	1	SORTIE	ENTREE
0	0	1	0	SORTIE	SORTIE	2	ENTREE	SORTIE
0	0	1	1	SORTIE	SORTIE	3	ENTREE	ENTREE
0	1	0	0	SORTIE	ENTREE	4	SORTIE	SORTIE
0	1	0	1	SORTIE	ENTREE	5	SORTIE	ENTREE
0	1	1	0	SORTIE	ENTREE	6	ENTREE	SORTIE
0	1	1	1	SORTIE	ENTREE	7	ENTREE	ENTREE
1	0	0	0	ENTREE	SORTIE	8	SORTIE	SORTIE
1	0	0	1	ENTREE	SORTIE	9	SORTIE	ENTREE
1	0	1	0	ENTREE	SORTIE	10	ENTREE	SORTIE
1	0	1	1	ENTREE	SORTIE	11	ENTREE	ENTREE
1	1	0	0	ENTREE	ENTREE	12	SORTIE	SORTIE
1	1	0	1	ENTREE	ENTREE	13	SORTIE	ENTREE
1	1	1	0	ENTREE	ENTREE	14	ENTREE	SORTIE
1	1	1	1	ENTREE	ENTREE	15	ENTREE	ENTREE

III.3 Répartition des E/S du MPF-1B

La figure 66 indique le champ d'adresses ENTREE/SORTIE.

Le mot de contrôle destiné au 8255 est 90H (fig 65, exemple 2). Le port «A» est programmé en «ENTREE» tandis que les PORTS «B» et «C» sont en «SORTIE» (voir schéma fig. 66).

a) PORT «A» (adresse 00)

— bit 0-5 : connectés aux six rangées de la matrice clavier. Le signal d'entrée est un niveau bas seulement quand une touche dans une colonne «activée» est enfoncée.

— bit 6 : connectée à la touche USER Key niveau «0» quand la touche est enfoncée.

— bit 7 : entrée magnétophone (écouteur)

b) PORT «B» (adresse 01)

Contrôle les afficheurs 7 segments et le point décimal. Le figure 67 indique la position de chaque segment de leur emplacement dans le port B. Tous les bits sont actifs au niveau haut.

c) PORT «C» (adresse 02)

Bits 0-5 : utilisés pour la sélection des 6 afficheurs et des 6 colonnes du clavier. Le bit «0» valide l'afficheur le plus à droite ; tandis que le bit 5 valide l'afficheur le plus à gauche.

Bit 6 : contrôle le MONITEUR. L'état de ce bit ne doit être modifié en aucun cas.

Bit 7 : sortie magnétophone (micro) ; cette sortie est aussi connectée au «haut-parleur» et à la LED verte. La LED est allumée quand la sortie est à «0».

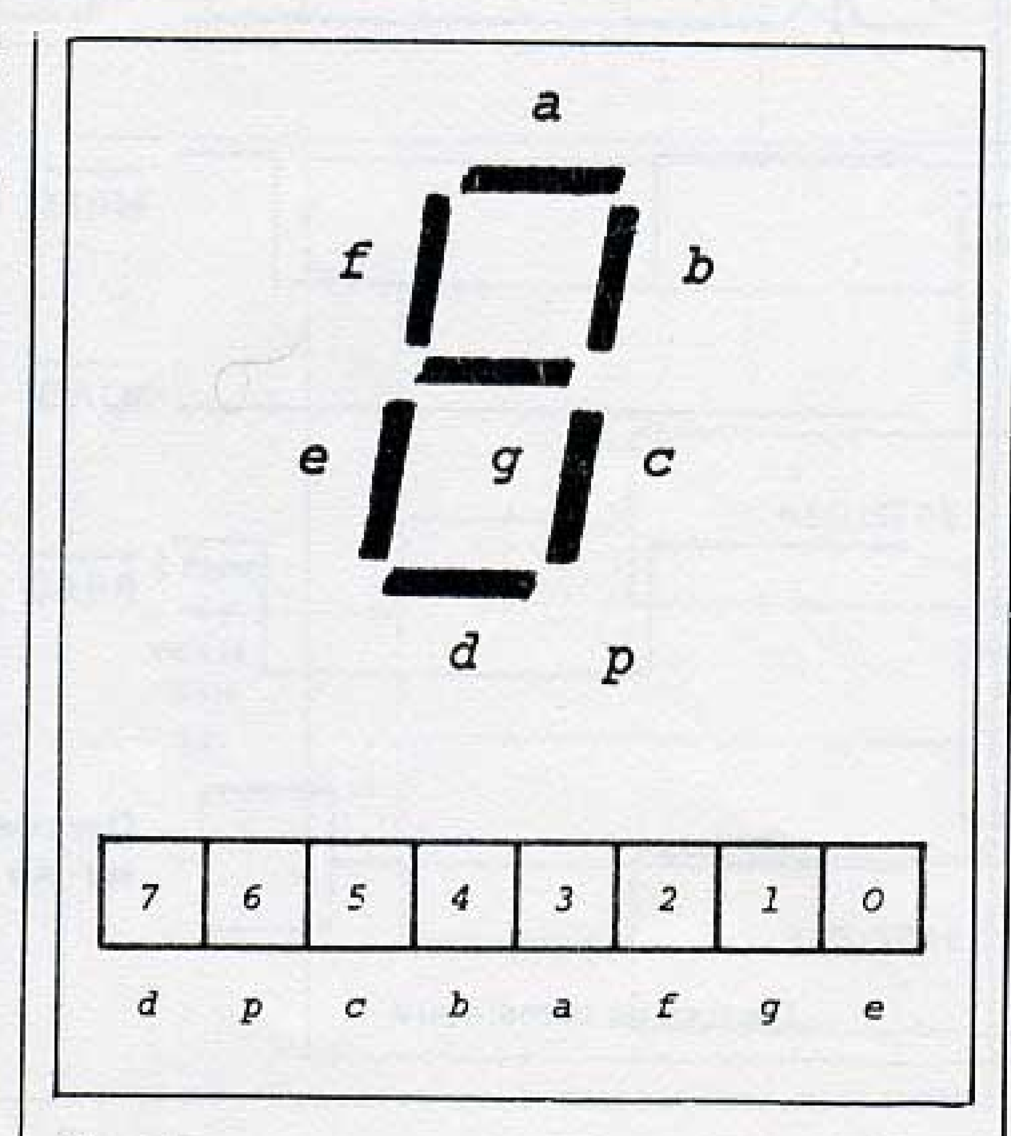


Fig. 67

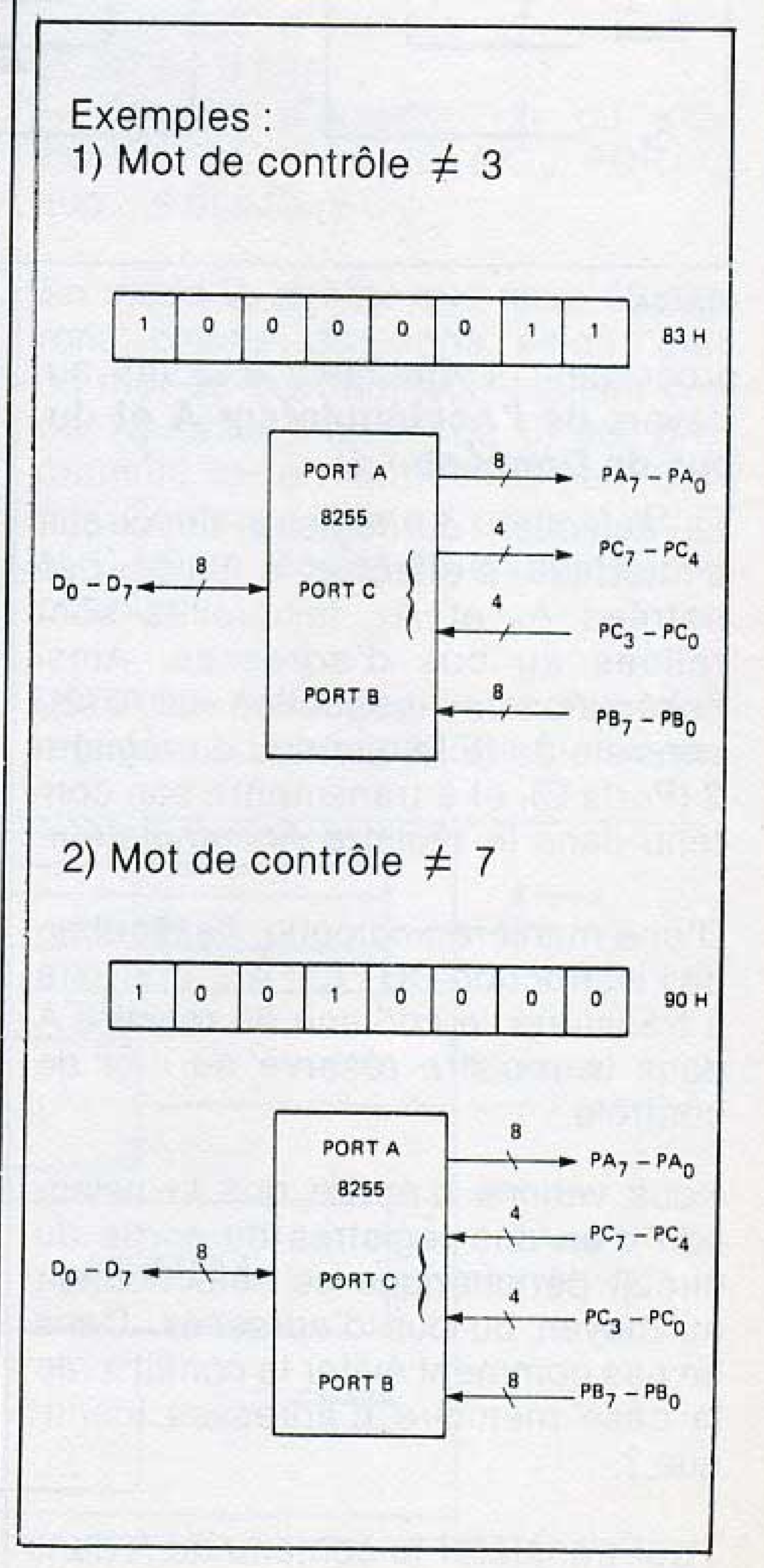


Fig. 66

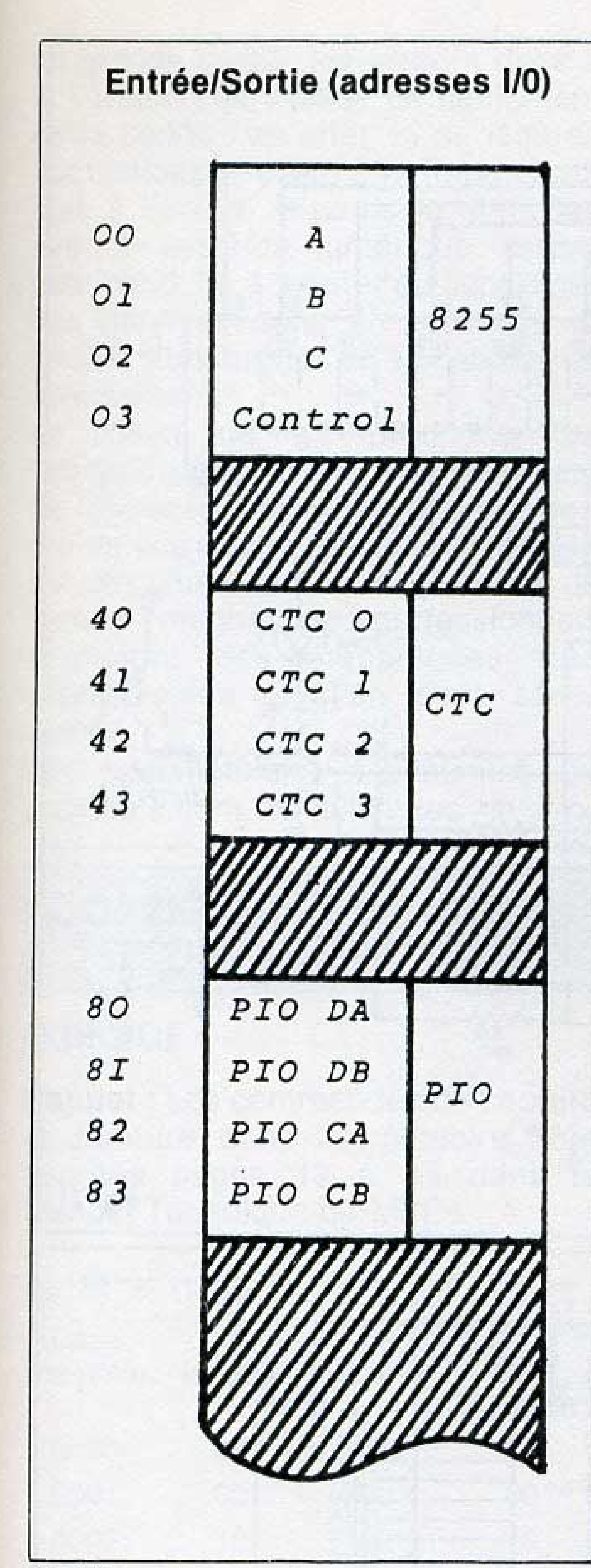


Fig. 68

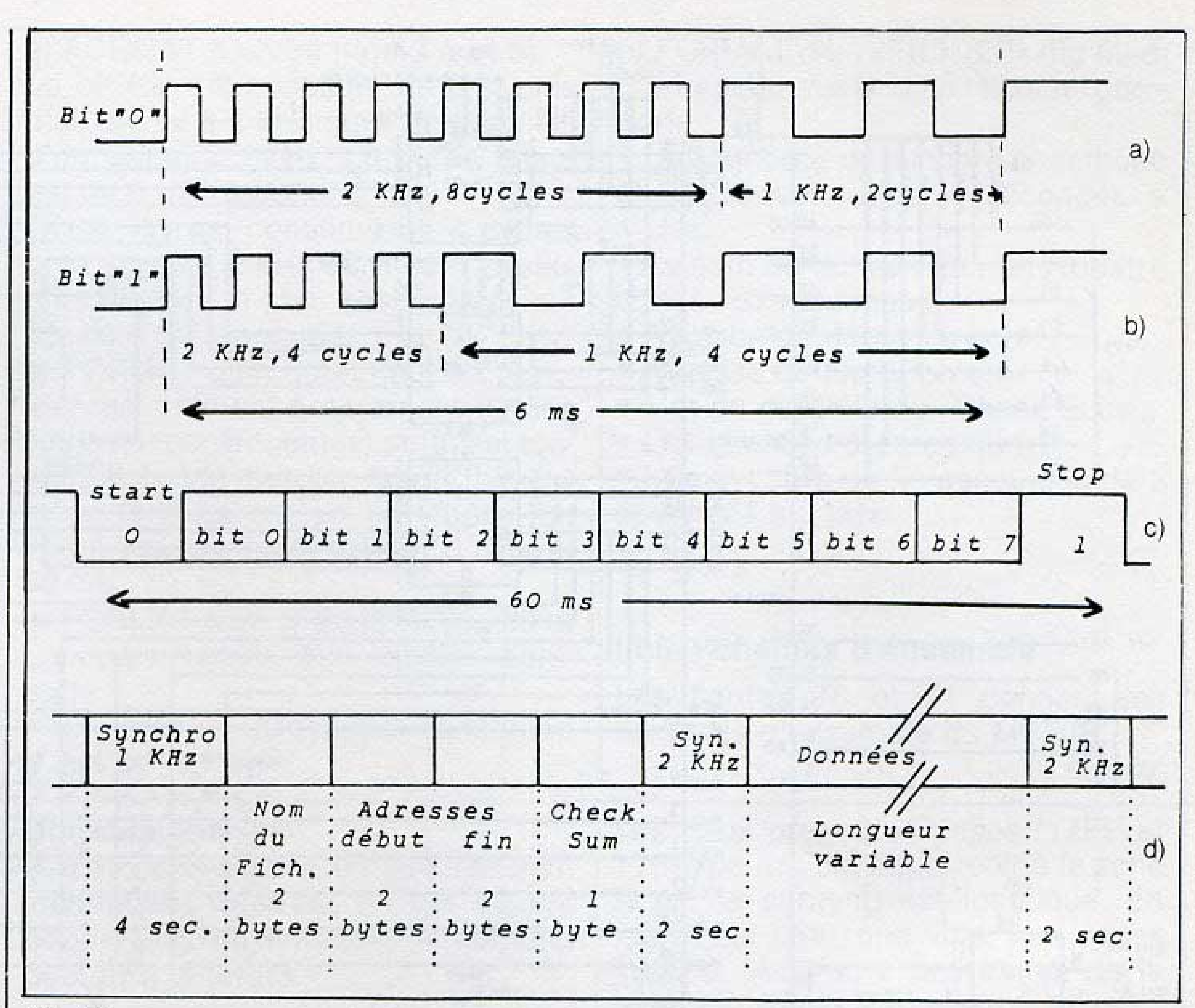


Fig. 70

III.4 Enregistrement sur cassette

La cassette magnétique constitue pour les systèmes informatiques destinés au grand public, une solution très économique pour disposer d'une «mémoire de masse» permanente.

Le MPF-1B possède une interface «Audio» connectée sur les ports du 8255 qui permet le raccordement à un enregistreur à cassettes du commerce. Deux prises «JACK» (Ø 3,5 mm) notées «EAR» et «MIC» permettent d'effectuer le branchement aux prises «ECOUTEUR» et «MICRO» du magnétocassette. La vitesse d'enregistrement et de restitution est de 160 bits/seconde.

Le détail de l'interface du point de vue hardware est indiqué sur la figure 68. Les broches «7» de A et C sont utilisés, l'un pour la lecture (Port A-7) l'autre pour l'enregistrement (Port C-7).

L'enregistrement «pur et simple» de «0» et «1» tels que les mots sont stockés dans la mémoire conduirait inévitablement à la relecture, à une catastrophe. Il faut notamment utiliser un «protocole» de transcodage, qui sera détaillé. Nous pouvons dire d'ores et déjà que le principal avantage de ce «traitement» est d'éliminer

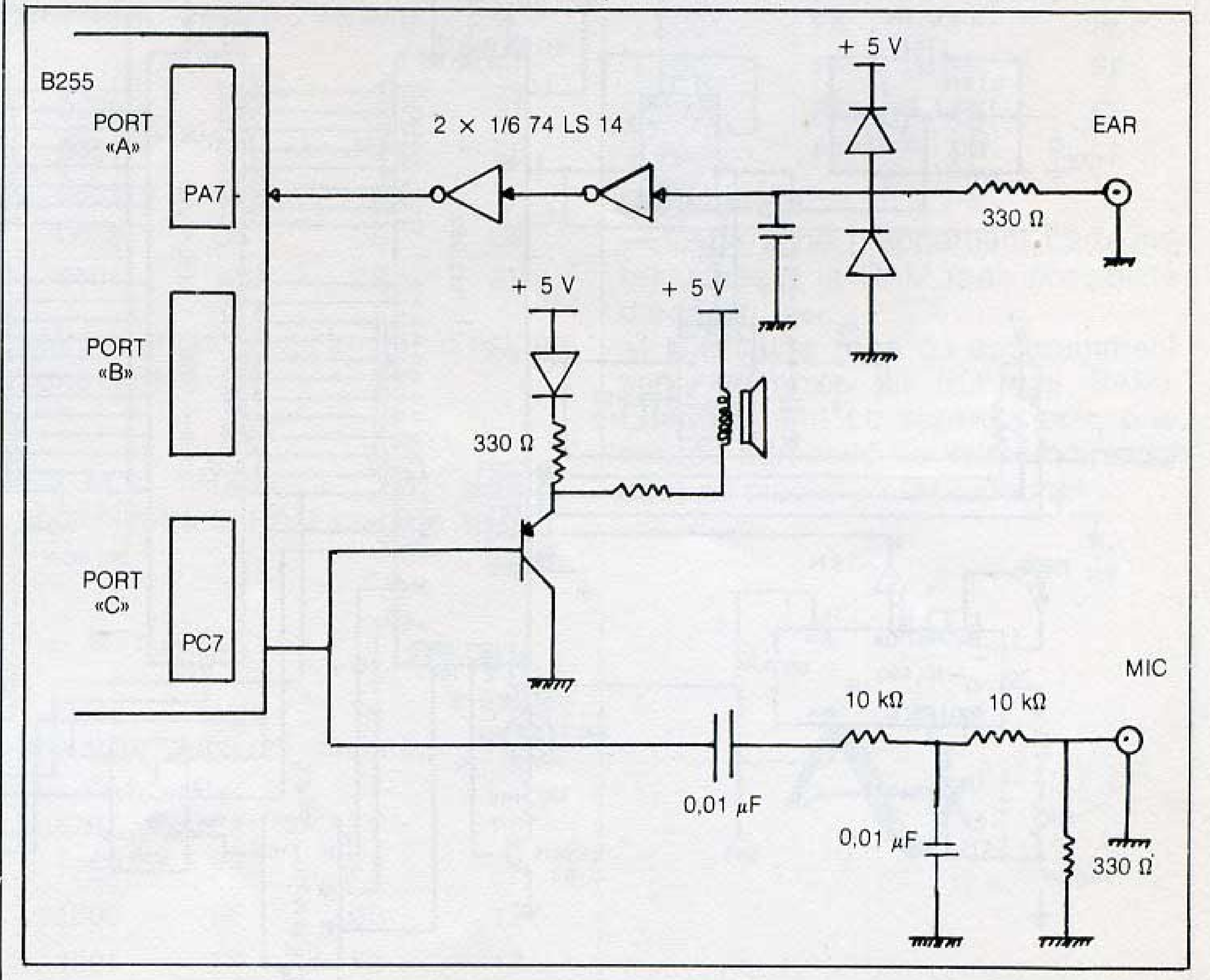


Fig. 69

Fig. 71

Fig. 72

en grande partie, les erreurs dues à la variation de vitesse de défilement de la bande ; en effet, si de légères fluctuations peuvent être imperceptibles à l'oreille elles ne le sont pas avec un système numérique : ce qui peut conduire à ajouter ou supprimer des données dans un message et encore transformer un «1» en «0» ou inversement.

La fidélité de la reproduction de l'enregistrement se fera au détriment de la vitesse, mais il ne faut pas perdre de vue que la mécanique utilisée est des plus sobre. Les systèmes de lecture magnétique professionnels emploient des mécaniques très sophistiquées et d'un coût assez élevé.

Les spécifications essentielles du protocole sont les suivantes :

a) FORMAT d'un bit (fig 69 a et b)

Le bit «0» est constitué de 8 cycles d'un signal à 2 kHz suivi de 2 cycles d'un signal à 1 kHz. La durée totale est de 6 millisecondes.

Le bit «1» est constitué de 4 cycles d'un signal à 2 kHz suivi de 4 cycles d'un signal à 1 kHz. La durée totale est de 6 millisecondes.

b) FORMAT d'un octet (fig 69 c)

L'enregistrement proprement dit des 8 bits qui constituent un octet est toujours précédé d'un bit «Start» (bit «0») et se termine par un bit «Stop» (bit «1»). La durée totale est de 10 x 6 soit 60 ms.

c) FORMAT d'un «FICHIER» (fig 69 d)

L'exregistrement d'un fichier comporte :

• Une amorce de synchro constituée d'une ss.

c) FORMAT d'un «FICHIER» (fig 69 d)

L'enregistrement d'un fichier comporte :

• Une amorce de synchro constituée d'une séquence de 4 secondes à 1 kHz.
• Le nom du fichier : 2 octets (quatre codes hexadécimaux).
• Adresse de début : 2 octets.
• Adresse de fin : 2 octets.
• Mot de contrôle ou «Cheek Sum».
• Les données à enregistrer.
• Une synchro de fin constituée de 2 secondes à 2 kHz.

III.5. Schémas d'ensemble

Les figures 70 et 71 donnent les schémas d'ensemble du MPF-IB.

(à suivre)

CORRIGÉS DES EXERCICES DU NUMÉRO 9

EXERCICE 1

Rappel : Les commandes de Lecture et Ecriture dans la mémoire sont décrites pages 13 à 15 dans le Manuel Technique du MPF-I.

1.a. M = 0000

Case mémoire	1re lecture	2e lecture	3e lecture après écriture
0000	06	06	06
0001	00	00	00
0002	10	10	10
0003	FE	FE	FE
0004	3E	3E	3E

Conclusion :

— Après chaque lecture le contenu est identique.

— Impossible d'ECRIRE dans cette zone. Les 5 cases appartiennent à la ROM.

1.b.M = 1900

Case mémoire	1re lecture	2e lecture	3e lecture après écriture
1900	0F	0F	10
1901	0F	0F	11
1902	F0	F0	12
1903	F0	F0	13
1904	0F	0F	14

Conclusions :

— Les deux premières lectures sont identiques : ceci est en fait « trompeur ». Sur votre matériel le contenu peut être différent.

— Possibilité d'écrire dans cette zone. Les 5 cases appartiennent à la RAM (mémoire vive).

1.c. M = 0600

Case mémoire	1re lecture	2e lecture	3e lecture après écriture
0600	21	21	21
0601	E6	E6	E6
0602	1F	1F	1F
0603	CB	CB	CB
0604	7E	7E	7E

Même remarque qu'en 1a : c'est une zone ROM.

1.d. M = 1A00
appartient à la RAM comme 1b.

1.e. M = 17FE

Case mémoire	1re lecture	2e lecture	3e lecture après écriture
17FE	FF	FF	FF
17FF	FF	FF	FF
1800	0F	0F	12
1801	F0	F0	13
1802	0F	0F	14

Les deux premières cases (17FE et 17FF) pourraient appartenir à la zone ROM, le contenu est identique. En fait, c'est une zone vide. Les cases 1800 à 1802 sont des cases de la mémoire RAM.

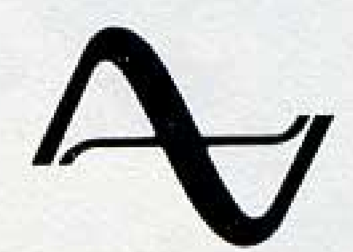
1.f. M = 2000

Case mémoire	1re lecture	2e lecture	3e lecture après écriture
2000	FF	FF	FF
2001	FF	FF	FF
2002	FF	FF	FF
2003	FF	FF	FF
2004	FF	FF	FF

Conclusion :

— Cette zone n'appartient certainement pas à la RAM (pas possibilité d'écrire).

— C'est une zone où apparemment sans mémoire (ni ROM ni RAM). Effectivement ce sont les cinq premières adresses de la mémoire qui peut se placer en U7 (option).



COURS PRATIQUE DE MICROPROCESSEUR AVEC LE MICROPROFESSOR MPF-IB

QUATRIEME PARTIE

Le hardware du MPF-IB (2)

SOMMAIRE

IV. LES MEMOIRES

1. Introduction

Nous avons vu dès la première partie (LED MICRO n° 9) que tout système à microprocesseur contient au moins deux types de mémoires. L'une d'elles est dite «mémoire morte», son contenu est **figé** et **non volatile,** elle renferme le programme permanent. Celui-ci peut être les algorithmes arithmétiques comme dans le cas d'une calculatrice ou le «MONITEUR» dans le système Microprofessor MPF-IB.

La seconde mémoire est du type **«vive» ou encore dite «mémoire de travail».** Elle est **volatile,** c'est-à-dire que son contenu disparaît dès qu'elle n'est plus alimentée. La mémoire vive stocke les données (ou DATA) dans le cas d'une calculatrice. Dans les systèmes de développement, l'utilisateur y introduit le programme à étudier ou à mettre au point. La raison essentielle est la très grande souplesse d'écriture et de lecture.

Sur la figure 54 (LED MICRO n° 11), le programme «MONITEUR» est situé entre les adresses 0000 et 0FFF (ROM ou EPROM) tandis que la mémoire RAM (6116) se trouve entre les adresses 1800H et 1FFFH.

Dans cette étude, nous ne présentons que les mémoires fabriquées sous forme de circuits intégrés, ce qui exclut les mémoires magnétiques (disques, bandes, par exemple) sauf toutefois les «mémoires à bulles» que nous évoquons pour information.

2. Organisation

Quel que soit le type de mémoire considérée («morte» ou «vive») et aussi la technologie employée (MOS, NMOS, CMOS, BIPOLAIRE, etc.), une mémoire se présente comme un grand damier (imaginez un jeu d'échecs ou de dames), où chaque case est une **cellule mémoire élémentaire,** c'est-à-dire qu'elle contient un bit qui est soit à «0» soit à «1». L'élément mémoire peut être un bistable (ou bascule), présence ou absence d'une diode, une capacité chargée ou déchargée, etc.

Le «damier» est composé de lignes et de colonnes (fig. 73). Par construction, le nombre de colonnes est 1, 4 ou 8. Le nombre de lignes est toujours une puissance de 2. Les plus petites ne comportent que 16 lignes (2^4) tandis que les plus grandes possèdent 65 536 lignes (2^{16}).

N'importe quelle **ligne N est lue individuellement.** L'opération consiste à connecter la ligne sélectionnée, et uniquement elle, avec les amplificateurs de sortie (fig. 73). Le «mot», l'octet dans le cas de la figure, est disponible en sortie de la mémoire. La lecture n'est jamais destructive, ce qui signifie que l'emplacement mémoire N contient toujours l'information initiale après cette opération.

Le nombre de colonnes peut se réduire à... un, ce qui ramène le mot de sortie à 1 seul bit. Il existe effectivement des mémoires de 65 536 lignes où chaque ligne ne comporte qu'une unique case. Ce type de configuration ne se rencontre que pour les

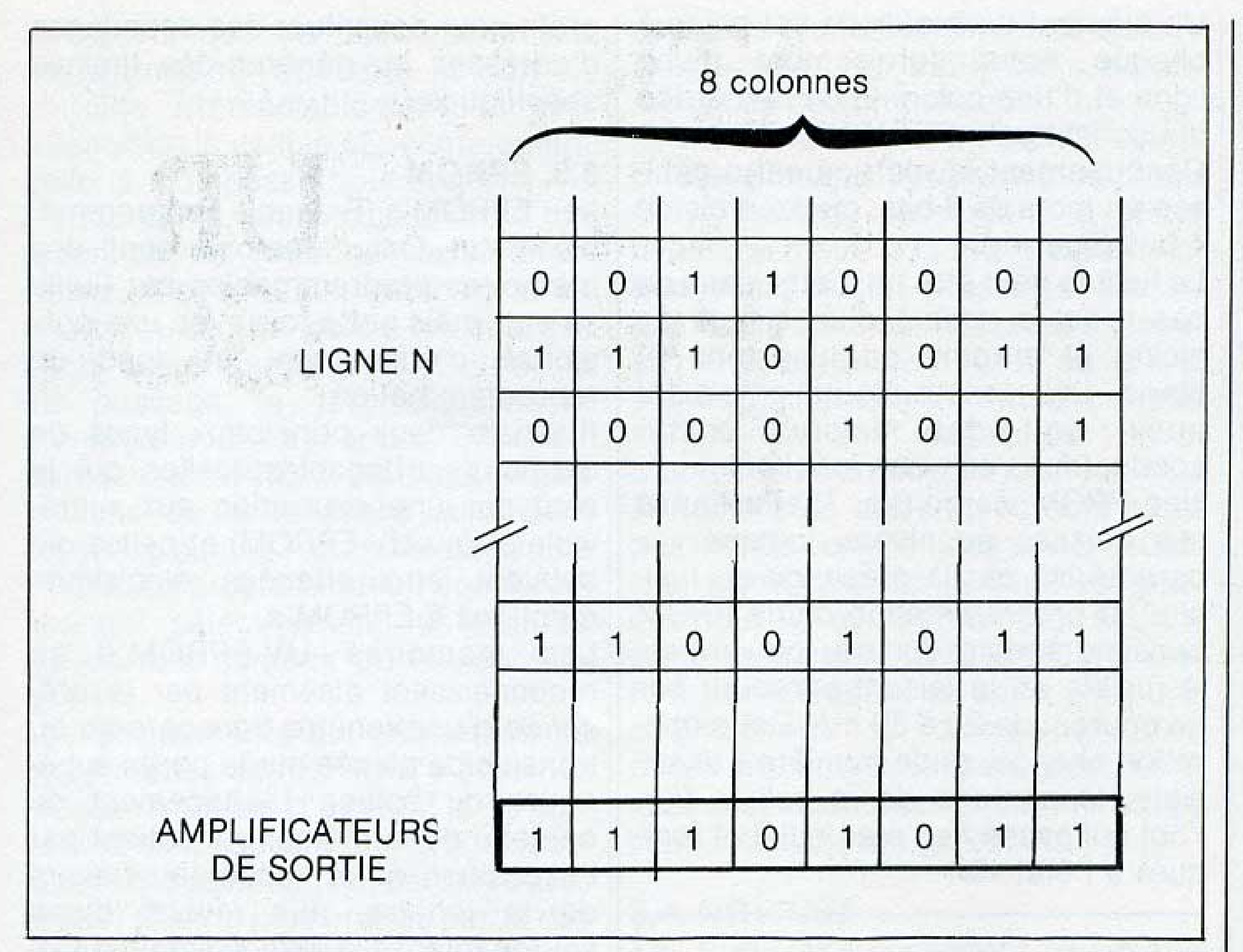


Fig. 73

mémoires vives. Dans ce cas, l'utilisateur place côte à côte autant de boitiers qu'il souhaite et il réalise ainsi des mots de longueur standard (8 bits) ou non standard, 13 bits par exemple.

Un autre avantage des mémoires 1 bit est leur grande capacité et leur faible prix de revient. Elles sont encapsulées dans des boîtiers «dip» 14 broches et se prêtent ainsi fort bien pour réaliser des «plans mémoires» relativement importants sans être trop onéreux.

Nous allons étudier les différents types de mémoires disponibles sous forme de «circuits intégrés», leur technologie et leurs caractéristiques essentielles. Pour permettre au lecteur de suivre efficacement cette présentation, nous commencerons par les mémoires les plus figées pour aller vers les mémoires les plus «souples».

3. Mémoires mortes

3.1. ROM

Les ROM's (Read Only Memory) signifie mémoire qui ne peut être que lue dans son utilisation normale. Ces mémoires sont programmées par le fabricant pendant le cycle de fabrication et leur contenu est **irrémédiablement figé.** Elles portent aussi le nom de ROM programmable par

Mémoires mortes

Type	Technologie	Organisation bits	Capacité Kbits	Tps d'accès ns	Consom. mW	Commentaires
ROM	MOS N MOS	8 (octet)	4 à 256 (32 Ko)	150 à 450	150 à 550	Produit figé, nécessite une quantité minimum de 10 000 pièces
	C MOS	8 (octet)	4 à 256 (32 Ko)	250 à 450	0,05 μW à 5 μW	Même remarque ; faible consommation
PROM	Bipolaire	8 (parfois 4)	0,256 à 64	20 à 60	500 à 900	Produit programmable. Consommation élevée. Temps d'accès faible
	MOS	8 (parfois 4)	0,256 à 64	400	130 à 800	Produit programmable. Consommation moyenne
EPROM	N MOS	8 (parfois 4)	4 à 256	150 à 450	100 à 800	Programmable et effaçable aux U.V.
	C MOS	8 (parfois 4)	32 à 256	200 à 300	0,5 à 40	
E EPROM	MOS	8 (parfois 16)	0,256 à 64	100 à 500	300 à 700	Effacement sélectif par octet. Temps d'effacement 10 ms. Nécessite une tension de 21 V
NOVRAM		1, 4, 8 ou 16	0,256 à 4	300	100/300	Combinaison d'une RAM et d'une E EPROM. Sauvegarde permanente sans aucune alimentation

Tableau I

masque. Le client fournit le programme à mémoriser (listing, disquette, bande magnétique, etc.) au constructeur qui génère à partir de ces documents l'un des masques de production qui fige le contenu «0» ou «1» de chaque cellule de la ROM au cours de sa fabrication.
Les coûts de développement et les outillages nécessaires pour réaliser une ROM sont très élevés, ce qui explique que les ROM's ne sont rentables que pour de très grosses quantités. Inversement le prix unitaire est faible. Ainsi toutes les calculatrices sont équipées d'une ROM. Le programme «MONITEUR» du MPF-IB est habituellement stocké aussi en ROM.
Les ROM's sont généralement organisées en octets et leur taille varie de 1 Koctet à 32 Koctets ce qui représente plus de 256 000 cellules sur une pastille de quelques mm² de silicium.
La technologie est généralement du type MOS, NMOS ou CMOS et la tension d'alimentation +5 volts.
Le tableau I indique les différents types de mémoires mortes.
Les progrès technologiques en cours permettent d'espérer la réalisation de ROM dont la capacité pourra atteindre 1 Mégabit soit 128 Koctets !...

3.2. PROM

Dans les applications courantes, en dehors des grandes séries, les quantités de mémoire à réaliser ne justifient pas l'emploi de ROM's. Les constructeurs ont développé de ce fait des **mémoires programmables par l'utilisateur,** ce sont des PROM's (Programmable Read Only Memory). Les PROM's présentent le même aspect d'irréversibilité que les ROM's, car une fois programmées il est impossible de modifier leur contenu.
Dans une PROM, chaque cellule élémentaire, par construction, est constituée d'un élément diode en série avec un fusible (fig. 74).

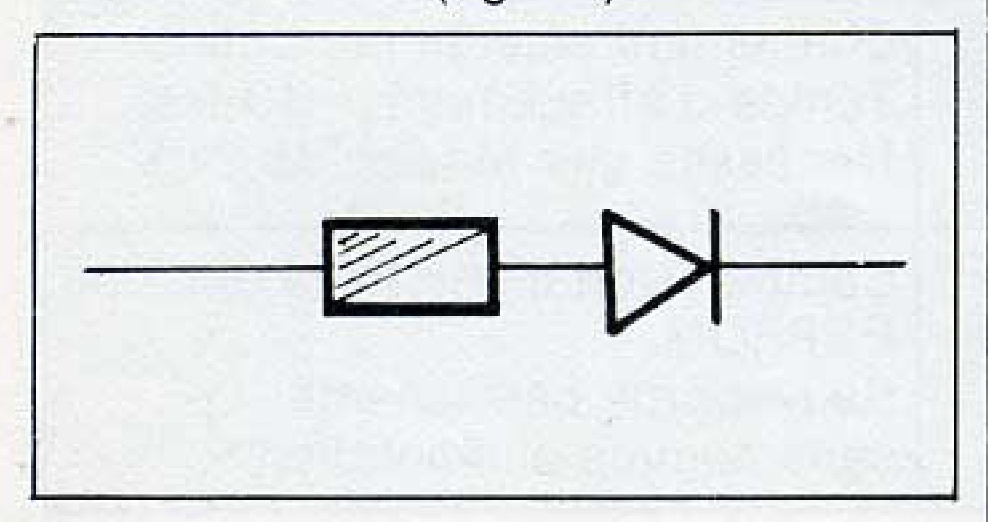


Fig. 74

Un élément fusible/diode est placé à chaque nœud (croisement d'une ligne et d'une colonne) de la matrice qui constitue la PROM (fig. 75a).
Généralement la matrice est organisée en mots de 8 bits, quelquefois de 4 bits (quartet).
Le fusible peut être un métal, dans ce cas il est constitué d'un alliage de nickel et chrome ou tungstène et titane. Les constructeurs emploient aussi soit des liaisons semi-conductrices soit une jonction.
Une PROM vierge (fig. 75a) a toutes ses cellules au niveau logique «1» caractérisé par la présence du fusible. La programmation d'une PROM consiste à «faire fondre» ou «brûler» le fusible en le faisant parcourir par un courant de 20 à 30 mA. Cette opération change, et de manière irréversible, le contenu de la cellule (fig. 75b) qui passe de l'état initial «1 logique» à l'état «0».

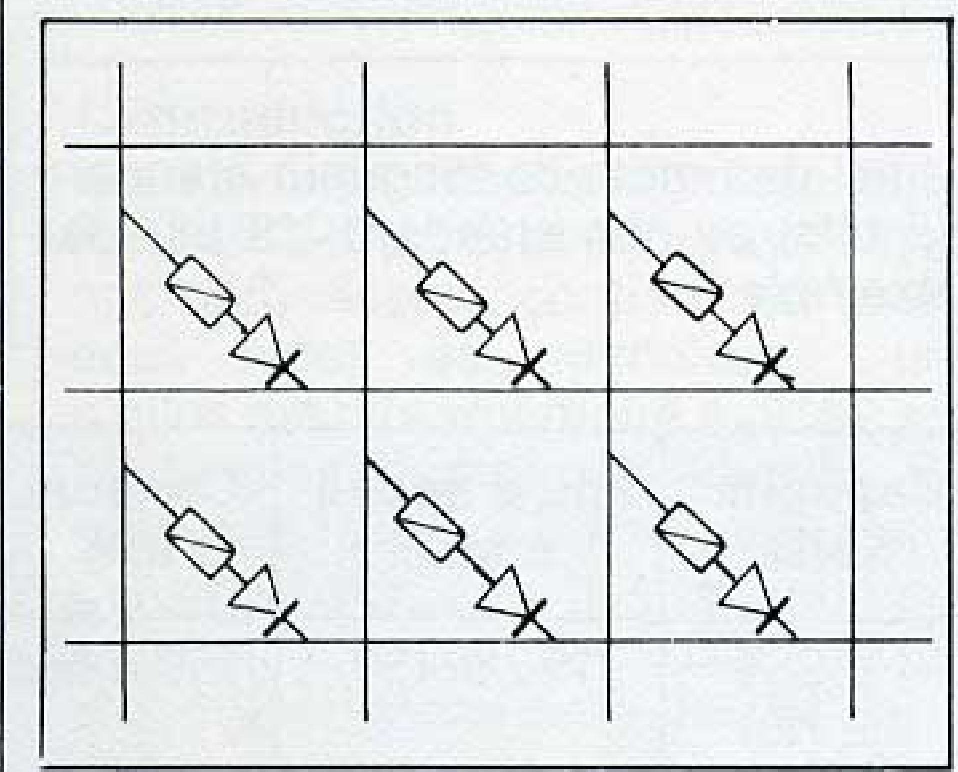


Fig. 75a

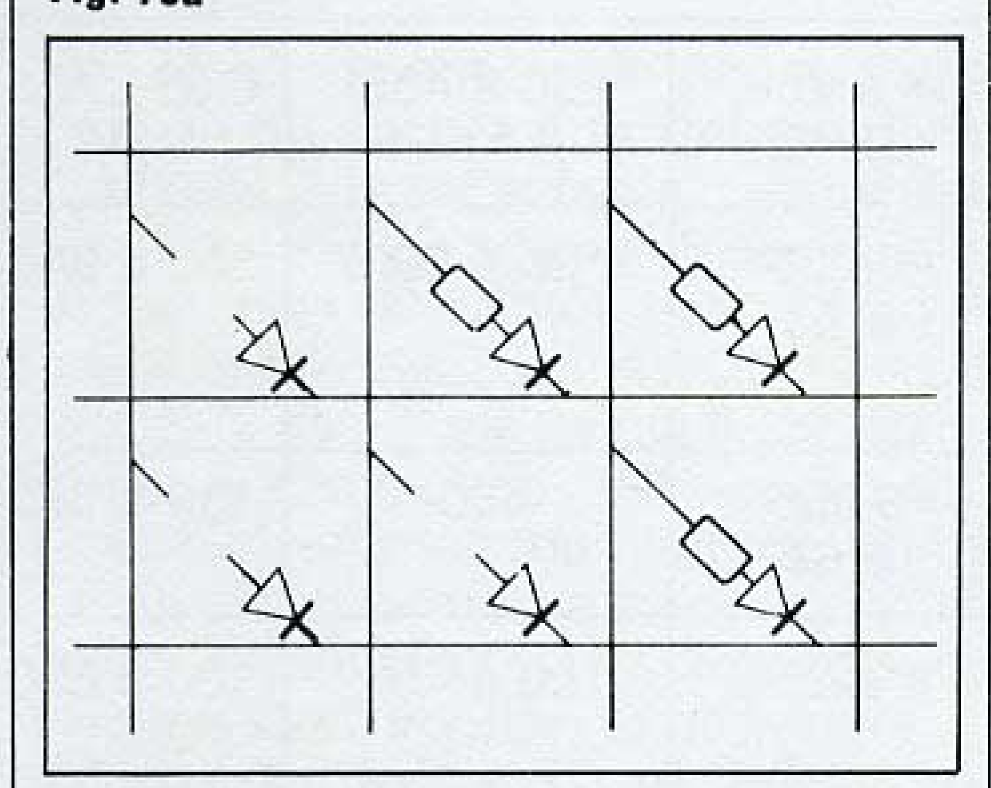


Fig. 75b

En examinant le tableau I, vous noterez que la capacité des PROM's est nettement plus faible que celle des ROM's, ceci est compensé par une grande flexibilité au niveau de la programmation. Il est cependant important de noter que le temps d'accès est plus court, quelques dizaines de nanosecondes avec des PROM's de type bipolaire. Cette dernière caractéristique est bien souvent mise à profit pour constituer des décodages d'adresses ou générer des timings spécifiques.

3.3. EPROM

Les EPROM's (Erasable Programmable Read Only Memory) sont des mémoires programmables par l'utilisateur, mais cette fois avec une possibilité d'effacement et donc de reprogrammation.
Il existe deux principaux types de mémoires effaçables, celles qui le sont par une exposition aux «ultra-violets» (ou UV-EPROM) et celles qui peuvent être effacées électriquement, les E-EPROM's.
Les mémoires UV-EPROM's se reconnaissent aisément par la présence d'une fenêtre transparente ou translucide placée sur la partie supérieure du boîtier. L'effacement du contenu de la mémoire s'obtient par l'exposition de la «puce» au travers de la fenêtre, aux rayons d'une source intense d'ultra-violets. La longueur d'onde de la lumière doit être de 2 537 Å ; la dose est intense, environ 10 W/s/cm² (watt-seconde par centimètre carré). Le temps d'effacement varie de 15 à 45 minutes. Celui-ci s'allonge avec le vieillissement du composant.
Lorsque une EPROM est programmée, il convient de masquer la fenêtre par une étiquette. Le rayonnement ambiant quel qu'il soit contient à plus ou moins forte dose des rayons ultra-violets mais leur intensité est suffisamment faible pour permettre de manipuler les EPROM's sans précaution particulière. Par contre une exposition permanente pourrait modifier le contenu mais après un temps relativement long.

Chaque cellule élémentaire, implantée sur la puce, est constituée par un transistor à effet de champ, dont la grille est flottante, c'est-à-dire isolée électriquement. La programmation d'une cellule (ou d'un bit) consiste à provoquer une accumulation de charges sur la grille du transistor : puisque celle-ci est isolée électriquement, il n'y a pas de chemin par lequel les charges peuvent s'écouler, tout au moins dans des conditions normales d'utilisation.
Au moment de la programmation, l'apport de charges sur une grille s'effectue par un effet d'avalanche en utilisant une tension élevée (25 volts ± 1 volt).

Il importe que ce phénomène soit parfaitement contrôlé sinon il entraîne irrémédiablement la destruction de la cellule et généralement celle du composant tout entier.
L'effacement, par exposition à un rayonnement intense aux ultraviolets, provoque un phénomène de photo-courant qui balaie les charges accumulées sur la grille. On notera au passage le fort déploiement d'énergie pour effacer une EPROM.
Lorsqu'une EPROM est vierge, tous les bits de la mémoire sont dans l'état logique «1» comme dans une PROM. La programmation consiste à changer sélectivement les cellules qui doivent contenir un niveau logique «0».
Les équipements qui permettent de programmer des EPROM's sont des programmateurs d'EPROM's. Ils possèdent bien souvent une grande souplesse d'adaptation qui permet de les utiliser pour la plus grande partie des composants disponibles sur le marché (2 Ko, 4 Ko ou 8 Ko).
Nous avons vu que les UV-EPROM's présentent une certaine souplesse de programmation par rapport aux PROM's et plus encore par rapport aux ROM's. Cependant elles présentent deux inconvénients, le premier est que l'effacement partiel ou localisé est impossible, d'autre part le composant doit être retiré du circuit pour être d'une part effacé et ensuite programmé. Les E-EPROM's suppriment ces deux handicaps.
Les E-EPROM's ou E^{2}PROM's (Electrically Erasable Programmable Read Only Memory) présentent le grand avantage d'être à la fois programmables mais aussi effaçables électriquement. De plus l'effacement peut se faire octet par octet ou d'une manière globale. L'écriture comme l'effacement sont réalisés par l'envoi d'une impulsion de 21 V. A noter que le nombre de cycles d'écriture n'est pas illimité comme dans une EPROM. Une E^{2}PROM 2816 de Intel est donnée pour 10^4 cycles par octet. Malgré cette restriction, les EEPROM's restent des éléments de choix pour des applications qui demandent des mises à jour périodiques (pompes à essence, distributeurs automatiques de billets dans les gares, etc.)

3.4. NOVRAM

Les E^{2}PROM's possèdent l'avantage de conserver en permanence leur contenu, mais par contre elles ont un nombre de cycles d'écriture limité et le temps d'effacement relativement long (10 ms). Certains constructeurs (General Electric, NCR, SGS...) ont développé des composants hybrides qui allient la rapidité de fonctionnement d'une RAM statique et le pouvoir de «stockage sans énergie» des E^{2}PROM's : ce sont des NOVRAM's. Ces composants sont constitués d'une RAM et d'une EEPROM (fig. 76). Lorsque le système est alimenté, seule la partie RAM fonctionne. Par contre, lorsque la tension disparaît, le contenu intégral de la RAM doit être transféré dans l'EEPROM. Cette sauvegarde contrôlée par le logiciel, ne nécessite que quelques dizaines de millisecondes et les capacités de filtrage du système sont dimensionnées de telle manière qu'elles fournissent l'énergie nécessaire pour cette opération. La recopie RAM$\rightarrow$E^{2}PROM à la coupure et la restitution (E^{2}PROM$\rightarrow$RAM) à la remise sous tension s'effectue automatiquement.
Les capacités des NOVRAM's sont encore limitées à quelques kilooctets, mais ce type de «mémoire bloc-note» trouve bien souvent son utilité dans des automatismes pour conserver des paramètres importants.

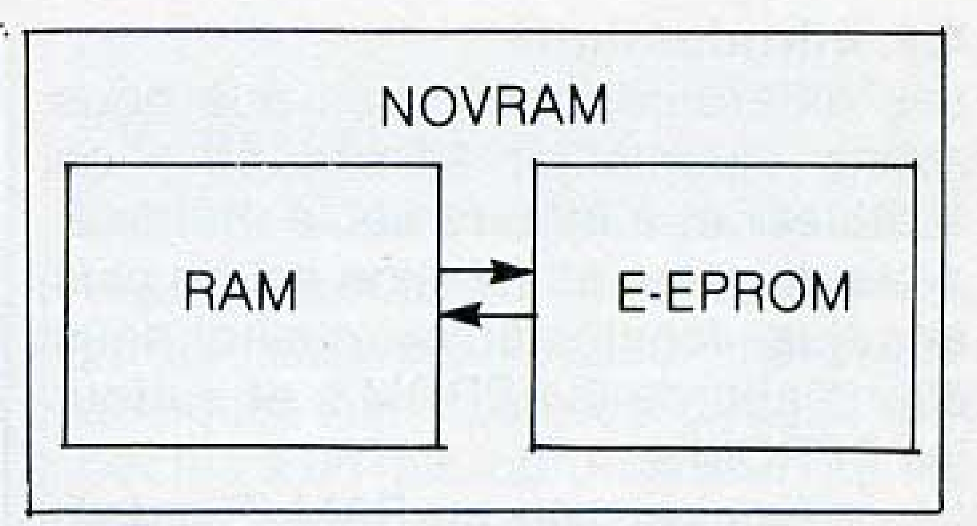


Fig. 76

4. Mémoires vives

Type	Technologie	Organisation bits	Capacité Kbits	Tps d'accès ns	Consom. mW	Commentaires
RAM dynamique	MOS N MOS	1, 4, 8	16 à 256	100 à 350	30 à 400	Ces mémoires doivent être périodiquement (3 ms) rafraîchies pour conserver leur contenu
RAM statique	MOS N MOS	1, 4, 8	4 à 64	150 à 400	150 à 600	Densité plus faible que les Ram dynamiques. Généralement 2 transistors par cellules
	C MOS	1, 4, 8	0,256 à 64	100 à 600	20 à 100	Faible consommation. Ce type de Ram se prête bien pour une sauvegarde par piles
	ECL	1, 4, 8	0,256 à 4	10 à 45	400 à 1 000	Ram avec des temps d'accès faible au détriment de la capacité (max. 4 Kbits) et une consommation élevée
	Bipolaire T.T.L.	1 ou 4	64 bits à 4 Kbits	33 à 50	175 à 500	Technologie bipolaire (T.T.L.) où chaque cellule est un bistable

Tableau II

4.1. Introduction

Les différentes mémoires que nous avons examinées sont dites «non volatiles» et à lecture seule même si cette dernière affirmation est de plus en plus «contestable» quand nous avons abordé les PROM's et surtout les EPROM's.

La «mémoire vive» ou RAM (Random Access Memory) est une mémoire dans laquelle l'information peut être à la fois lue ou écrite avec la même rapidité uniquement sous l'action de commandes spécifiques contrôlées par le microprocesseur. Dans une calculatrice, les données introduites au clavier ainsi que le résultat sont stockés en mémoire vive.

L'organisation d'une RAM est identique à celle des ROM's. Chaque ligne peut être «adressée» individuellement. Après avoir sélectionné la cellule N, on peut sauter à «N + 10» ou «N – 15». L'accès de la mémoire, aussi bien en écriture qu'en lecture est dit «aléatoire», d'où le nom donné par opposition aux mémoires magnétiques (bandes, disquettes, etc.) où l'accès est séquentiel.

On distingue deux types essentiels de mémoires :

— les mémoires RAM dynamiques
— les mémoires RAM statiques.

4.2. RAM dynamique

Les RAM's dynamiques ne sont réalisées qu'en technologie MOS car la **cellule mémoire élémentaire** est constituée par la **capacité** que constitue l'entrée (GATE) avec le substrat (fig. 77).

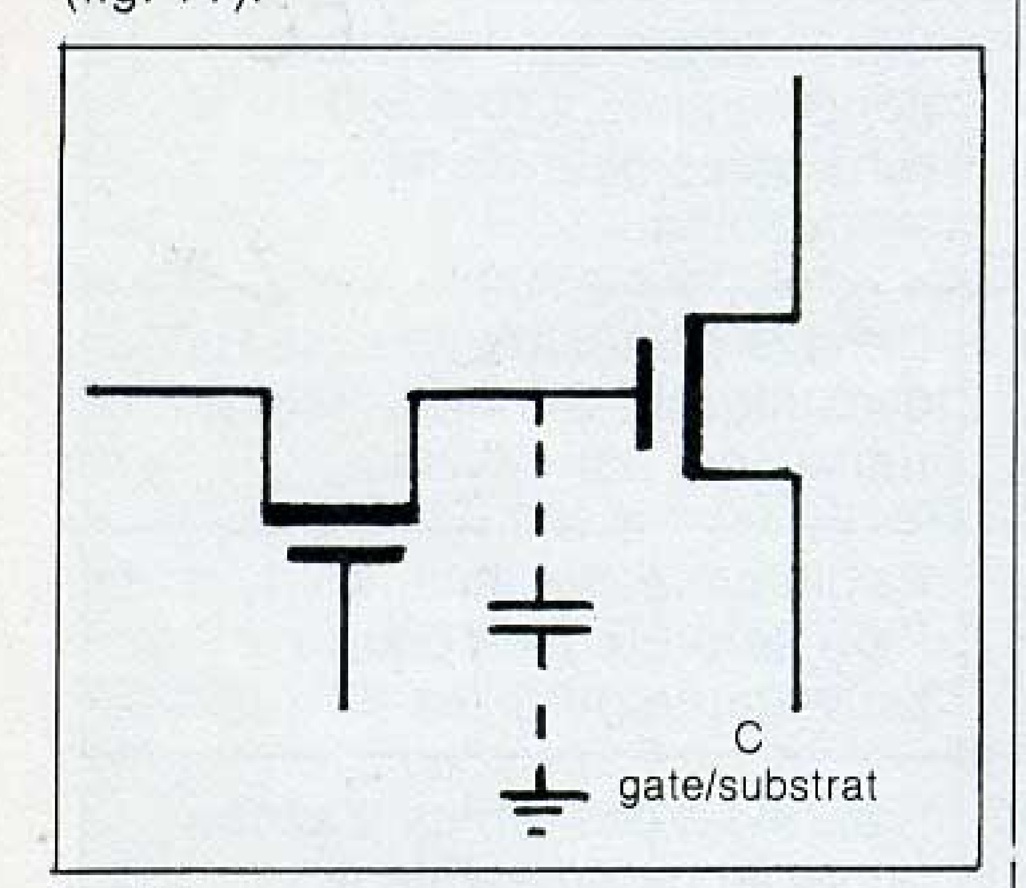


Fig. 77

La valeur du bit de chaque cellule «0» ou «1» est représentée par l'absence ou la présence de charges contenues dans le condensateur C, qui compte tenu de la haute impédance de la «gate» conserve un état stable durant un certain laps de temps : généralement de 8 à 10 ms, ce qui est peu. L'effet «mémoire» doit être prolongé par une opération de «rafraîchissement» (Refresh) périodique environ toutes les 2 à 3 millisecondes.

Le «rafraîchissement» consiste à lire le contenu de chaque cellule de la mémoire et de le réinscrire au même emplacement ; en quelque sorte «refaire le plein» des capacités qui étaient chargées avant qu'elles n'atteignent le seuil bas et maintenir déchargées les autres. Le rafraîchissement compense les méfaits des courants de fuite. C'est cette opération de rafraîchissement qui a conduit les constructeurs à appeler «dynamique» ce type de mémoire.

Le temps nécessaire au rafraîchissement des mémoires dynamiques peut sembler une contrainte avec le risque de réduire les performances d'un système. En réalité il n'en est rien, tout au moins avec les microprocesseurs haute performance, comme le Z80^R qui génèrent automatiquement les signaux de commande (RFSH) pour le rafraîchissement pendant le cycle d'exécution des instructions.

Les RAM's dynamiques, compte tenu de la simplicité de la cellule élémentaire, permettent d'atteindre des densités relativement importantes (voir tableau II) sur des surfaces restreintes. Actuellement des RAM's 256 kbits (256 K × 1) sont fabriquées par la plupart des constructeurs.

4.3. RAM statique

Alors que la cellule d'une RAM statique ne comporte qu'un seul transistor (toujours MOS), la cellule élémentaire d'une RAM statique est constituée par une cellule bistable, c'est-à-dire au moins deux transistors.

Une cellule bistable ou bistable est un circuit symétrique (fig. 78) qui possède deux états stables. En désignant par Q et Q' les deux sorties, les états possibles sont :

$$Q = 0 \text{ et } Q' = 1$$
$$\text{ou } Q = 1 \text{ et } Q' = 0$$

A la mise sous tension, si une parfaite symétrie de construction est respectée, la sortie Q prend indifféremment la valeur «1» ou «0». C'est pourquoi le contenu d'une RAM est aléatoire lors de l'initialisation. Des défauts de symétrie peuvent favoriser un état plutôt que l'autre à la mise sous tension, mais le contenu reste cependant toujours incertain avant une première écriture.

Du point de vue fonctionnel, chaque ligne peut être écrite et lue avec des temps d'accès de l'ordre de 100 à 300 nanosecondes.

L'organisation de ces mémoires sont de 1, 4 ou 8 bits. Les plus fortes densités sont obtenues avec l'emploi de la technologie MOS, comme pour les RAM's dynamiques. La capacité maximale est cependant 16 fois plus faible (voir tableau II) : ceci s'explique quand on compare les cellules élémentaires respectives (figures 77 et 78).

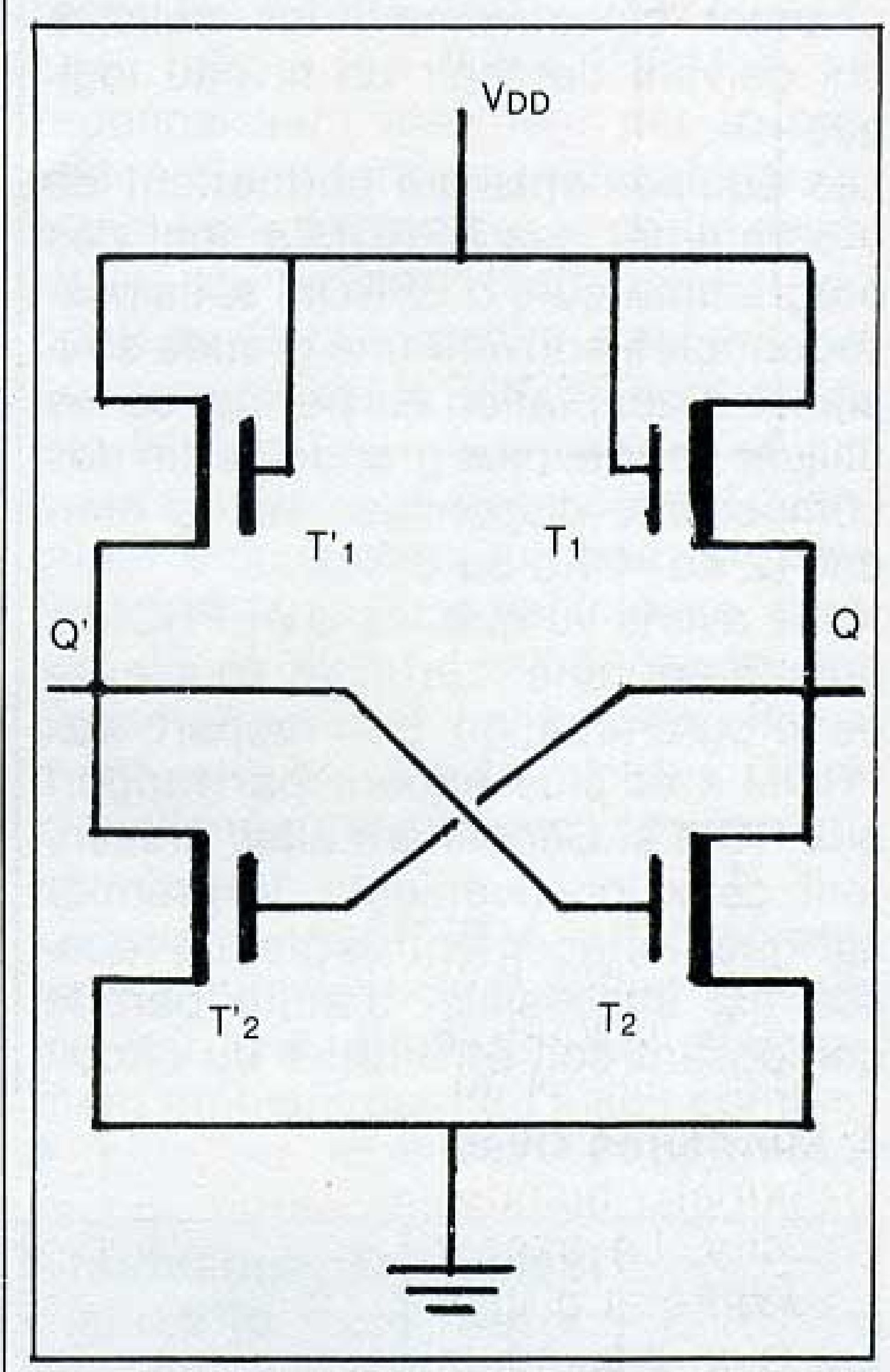


Fig. 78

Lorsque des temps d'accès de quelques dizaines de nanosecondes tant à la lecture qu'à l'écriture sont nécessaires, l'emploi de la technologie bipolaire est indispensable soit type T.T.L. ou mieux E.C.L. La plus haute vitesse est atteinte avec cette dernière (Emitted Coupled Logic) qui est une technique dans laquelle les **transistors fonctionnent en mode non saturé** ce qui n'est plus le cas de la T.T.L. L'accroissement de vitesse s'effectue au détriment de la consommation et de la densité qui ne dépasse pas 4 kilobits soit 16 fois moins que les RAM's statiques MOS.

4.4. RAM statique «Zero Power ^R +»

Pour terminer ce tour d'horizon sur les mémoires, nous présentons un circuit récent proposé par Mostek

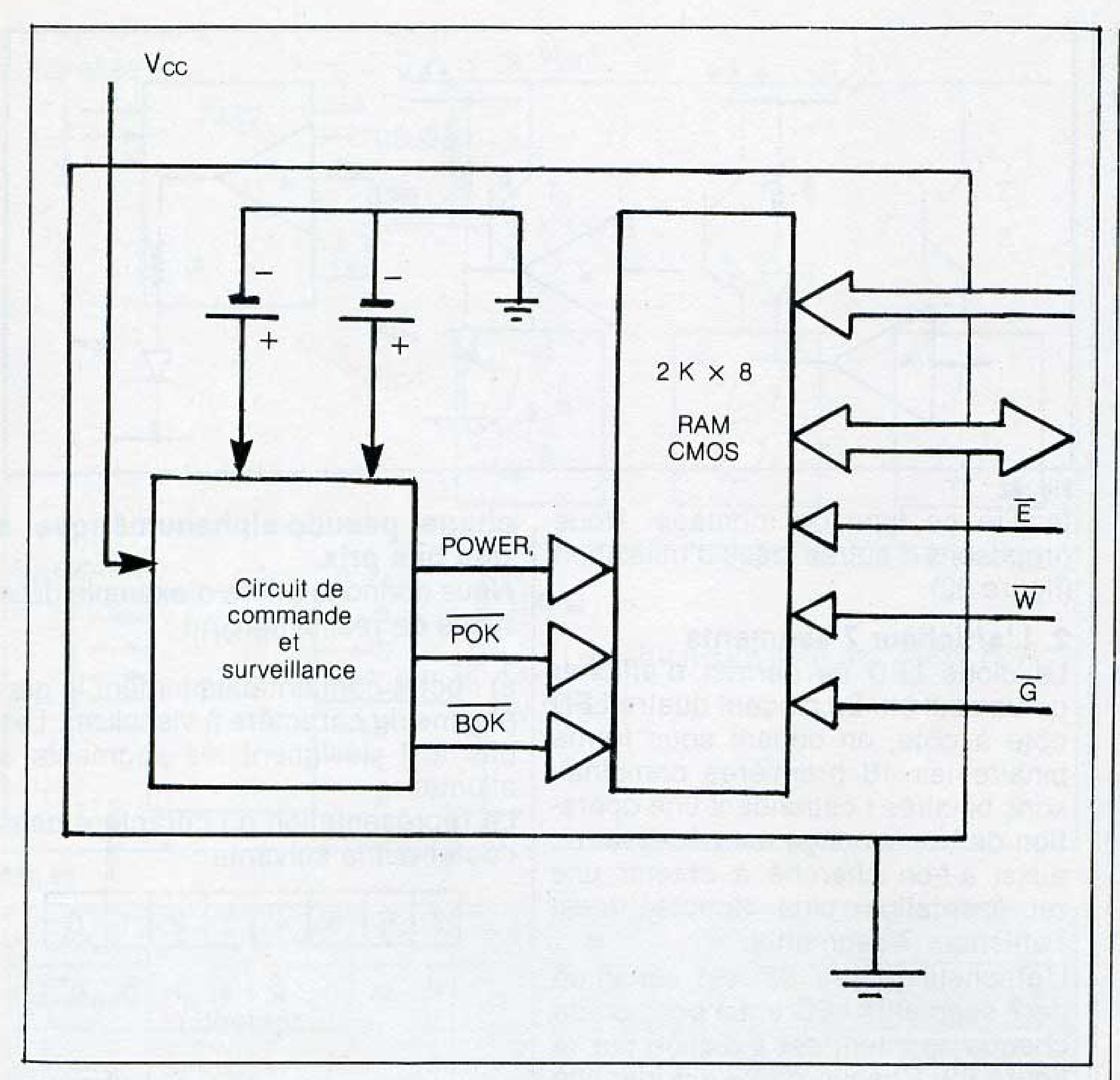


Fig. 79

intitulé «Zero Power». Comme nous l'avons indiqué dans le numéro précédent, les RAM CMOS possèdent deux modes de fonctionnement. L'un «normal» lorsque la tension est de 5 volts. Le second mode dit «stand by» ou état d'«hibernation» avec une tension d'alimentation plus faible (de l'ordre de 2 à 3 volts) et un courant de rétention de quelques micro-ampères. Dans cet état **le contenu de la mémoire est conservé,** mais les opérations **de lecture ou d'écriture sont impossibles.**

Le concept «Zero Power» de Mostek consiste à encapsuler dans un unique boîtier DIP, une RAM 2 K×8, 2 piles au lithium et un circuit de commande (fig. 79).

Les piles au lithium ne fournissent la tension de rétention à la RAM que lorsque la tension V_{CC} est au dessous d'un seuil (3 volts environ), le courant est de l'ordre de 0,3 μA. L'espérance de vie du produit est de l'ordre de 5 à 10 ans dans des conditions d'emploi normales.

5. Mémoire à bulles

Les mémoires à bulles développées par Intel ou Sagem permettent d'obtenir sur une surface faible des tailles de mémoire de 1 Megabit ou 4 Megabits. L'électronique de commande à mettre en place est relativement complexe et le contrôle s'apparente plus aux techniques DMA (Direct Memory Access) qu'à l'accès aléatoire comme nous l'avons vu jusqu'à présent.

6. Conclusion

Les tableaux I et II indiquent la situation des mémoires disponibles à ce jour. Il est bon de savoir que les constructeurs investissent des sommes très importantes pour améliorer les performances de ce type de composant (densité, temps d'accès, réduction de la consommation, etc.) car le marché est énorme et ne cesse de s'accroître.

En rédigeant cet article, j'ai reçu une documentation technique de A.M.D. (Advanced Micro Devices) qui commercialise, depuis février 84, une UV-EPROM, la Am 27512 d'une capacité de 65 535 octets soit 524 288 cellules mémoires !!!

V. INTERFACAGE

Nous présentons dans cette dernière partie relative à l'aspect «hardware» quelques exemples de circuits connectés à un microprocesseur. En réalité ils seront de préférence reliés à **un circuit périphérique d'Entrée-Sortie** (comme le 8255 ou le PIO-Z80) ce qui permet d'en connecter un plus grand nombre (le nombre de circuits périphériques adressables avec le Z80 est de 256).

1. L'afficheur à L.E.D.

La LED (Light Emitting Diode) est une diode au phosphore arséniure de Gallium (Ga As P) ou encore au phosphore de Gallium qui a la particularité d'émettre une lumière dans le spectre visible quand elle est traversée par un courant I_F suffisant.

Electriquement, la LED doit être considérée comme une diode avec une tension directe de l'ordre de 1,6 volts quand elle est traversée par un courant d'une vingtaine de milli-ampères ce qui correspond à une luminance de 700 (Ft — L) (Ft — L : Foot Lambert).

Lorsque l'ensemble est alimenté par une tension de 5 volts (fig. 80), la détermination de la résistance R à placer en série avec la diode est telle que :

$$R = \frac{V_{CC} - V_F}{I_F} = \frac{5 - 1{,}6}{0{,}020} = 170\Omega$$

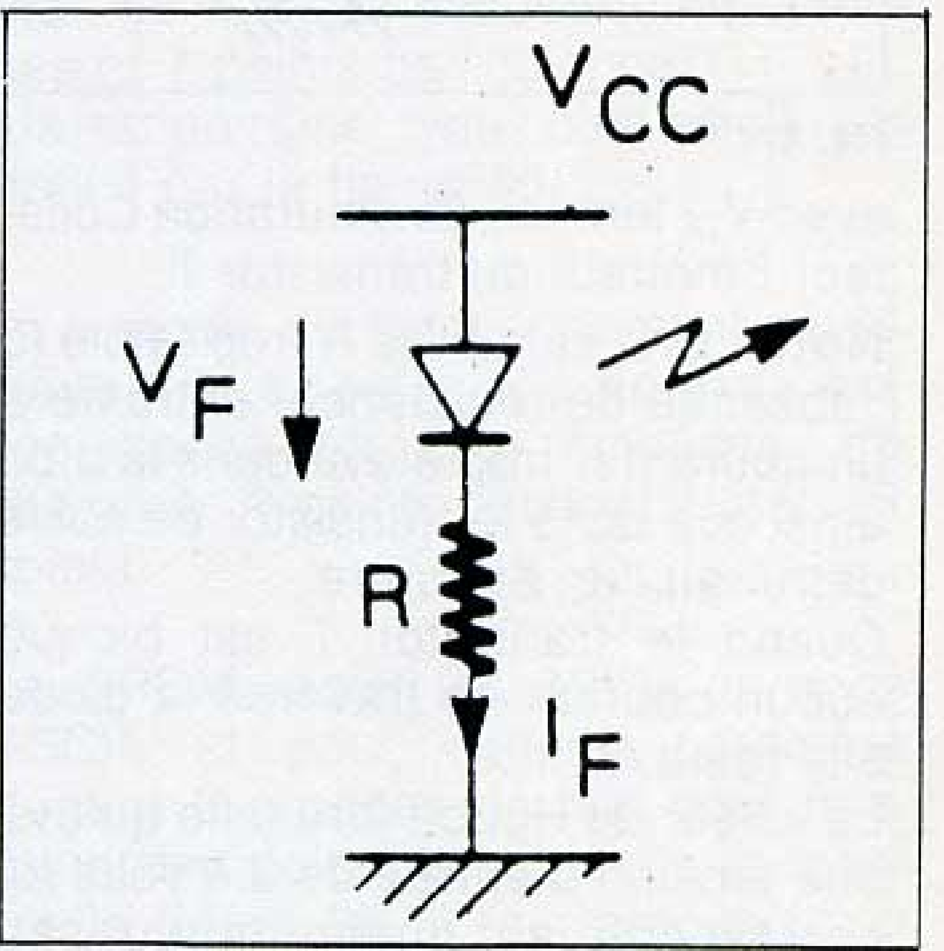


Fig. 80

La diode LED grâce à sa souplesse d'emploi (faible coût, faible consommation), est largement employée pour visualiser une **donnée binaire,** l'état d'une bascule ou tout autre information unitaire.

Le circuit que nous avons représenté figure 80 n'est pas utilisable très souvent tel quel si ce n'est pour indiquer la présence de l'alimentation V_{CC} par exemple. Les sorties d'un microprocesseur ne permettent pas de fournir un courant aussi important que celui demandé (nous rappelons que le courant maximum est de 1,9 mA). Il faut donc placer entre la sortie et la LED un circuit d'interface de puissance, généralement constitué par un transistor ou mieux un circuit intégré dont l'étage de sortie est un transistor en collecteur ouvert (par exemple SN 7407).

Dans ce cas la détermination de la résistance R (figure 81) s'obtient par la relation suivante :

$$R = \frac{V_{CC} - (V_1 + V_{CE})}{I_F}$$

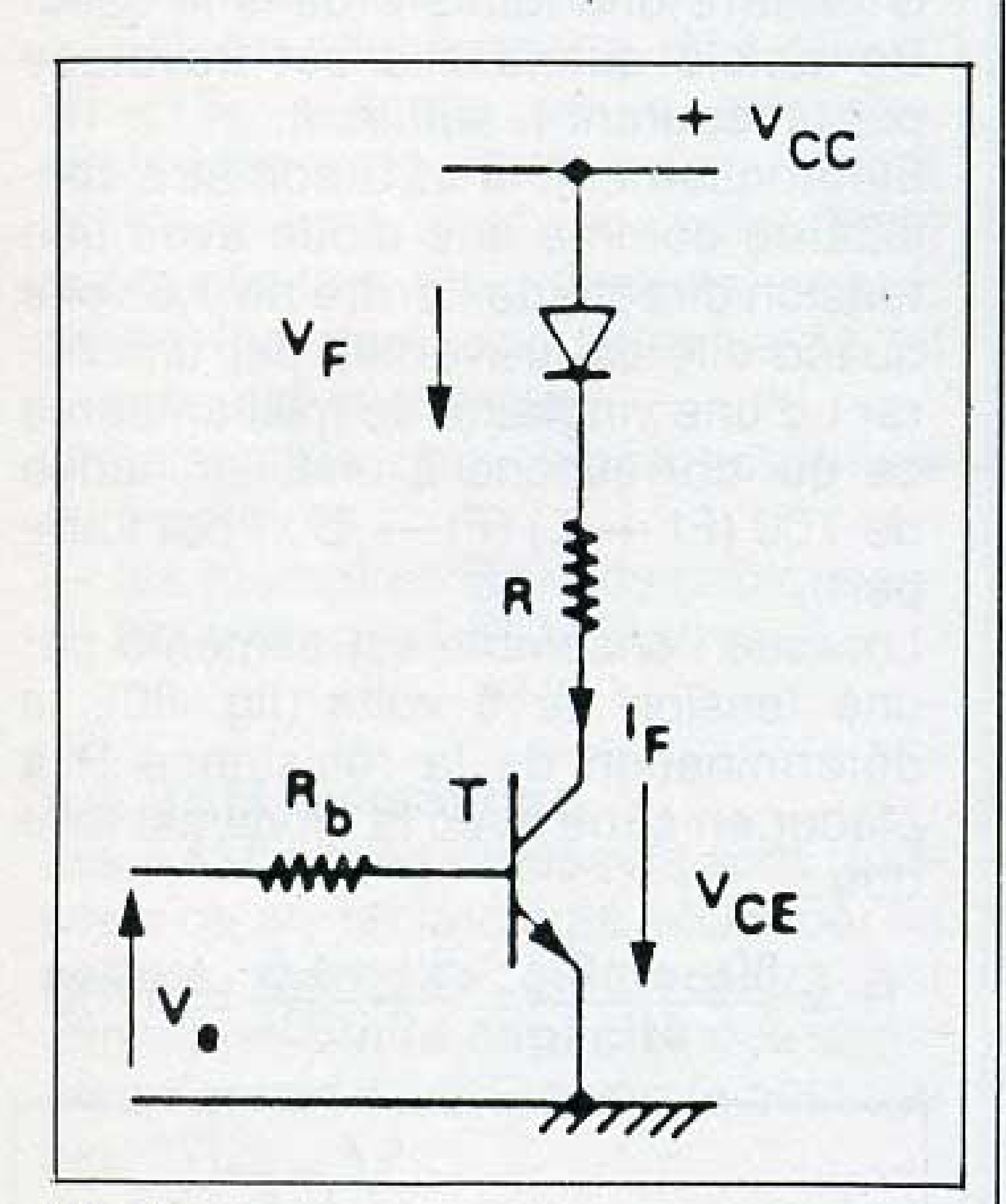


Fig. 81

avec V_{CE} tension de saturation Collecteur Emetteur du transistor T.

Nota : Une valeur de R trop faible (ou l'absence de résistance) entrainerait un courant I_F trop élevé dans la diode ainsi que dans le transistor ce qui les détruirait l'un et l'autre.

Quand le transistor T est bloqué : aucun courant ne traverse la diode : elle reste éteinte.

La valeur de R_b doit être telle qu'avec une tension d'entrée de 2,4 volts (qui correspond au minimum du seuil «haut» logique) le transistor T soit saturé. Dans ces conditions, la L.E.D. s'illumine quand un niveau 1 logique est placé à l'entrée du montage.

La plupart des sorties d'un microprocesseur sont directement compatibles avec les entrées T.T.L. ce qui

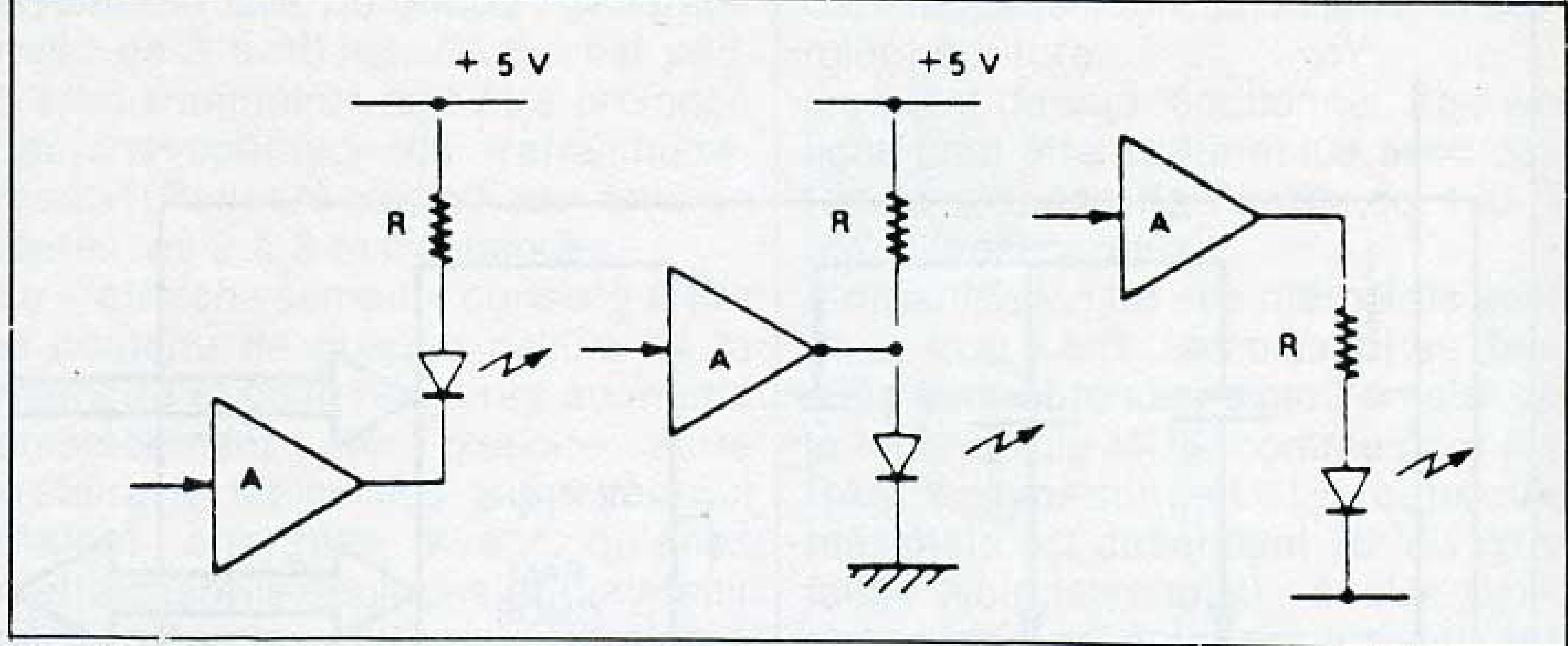


Fig. 82

facilite ce type de montage. Nous proposons d'autres idées d'utilisation (figure 82).

2. L'afficheur 7 segments

La diode LED ne permet d'afficher qu'un seul bit. En plaçant quatre LED côte à côte, on obtient sous forme binaire les 16 premières combinaisons binaires : cependant une opération de transcodage est nécessaire ; aussi a-t-on cherché à obtenir une représentation plus directe, c'est l'afficheur 7 segments.

L'afficheur (figure 83) est constitué de 7 segments LED's. La position de chaque segment est indiquée par la figure 83. Chacun d'eux est identifié

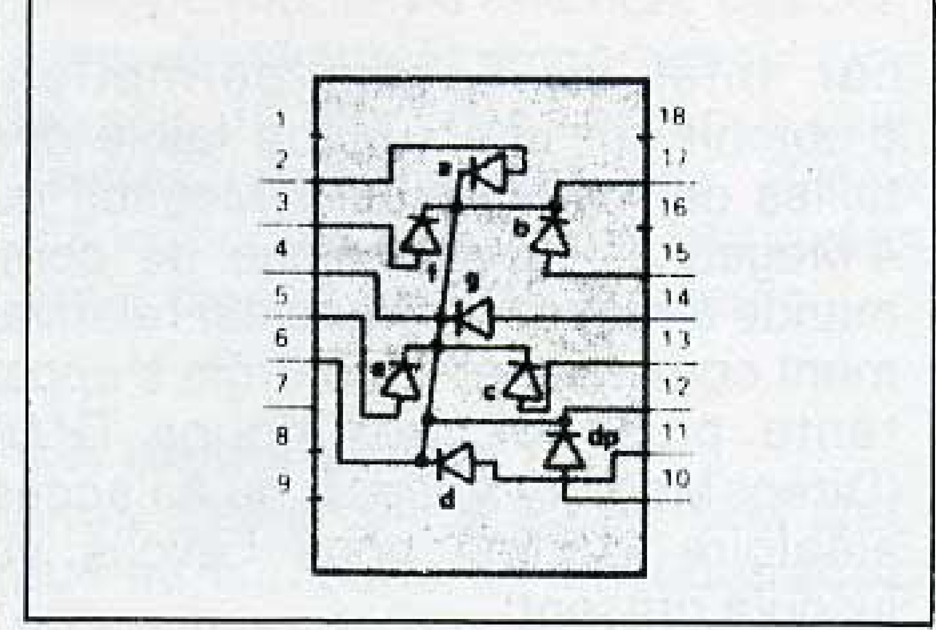


Fig. 83

par une lettre minuscule a, b, c, d, e, f et g.

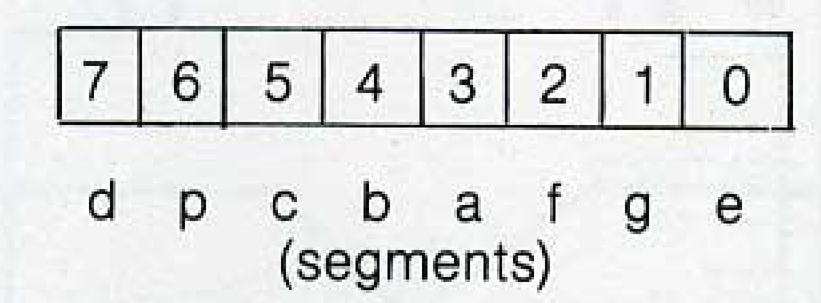
Avec ces 7 segments les chiffres de 0 à 9 et quelques lettres de l'alphabet peuvent être représentées. Ce dispositif permet de **visualiser sans ambiguïté les 16 caractères employés dans le code hexadécimal.** Il est couramment employé dans les systèmes d'initiation qui ne demandent que des messages numériques et un alphabet restreint.

Etant donné que toutes les combinaisons possibles ne sont pas toutes employées, pour certaines applications, et compte tenu de la souplesse d'emploi, le reste de l'alphabet et quelques signes sont réalisés tant bien que mal. On obtient ainsi **un affichage pseudo-alphanumérique à très bas prix.**

Nous donnons à titre d'exemple deux types de réalisation.

a) l'octet contient uniquement le graphisme du caractère à visualiser. Les bits à 1 désignent les segments à allumer.

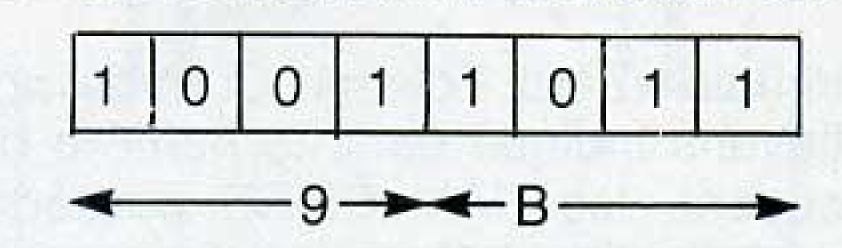
La représentation du caractère dans l'octet est la suivante :



7	6	5	4	3	2	1	0
d	p	c	b	a	f	g	e

(segments)

Le représentation du nombre 2 est :



1	0	0	1	1	0	1	1

Le code de 2 est 9 B en hexadécimal ou 1001 1011 en binaire.

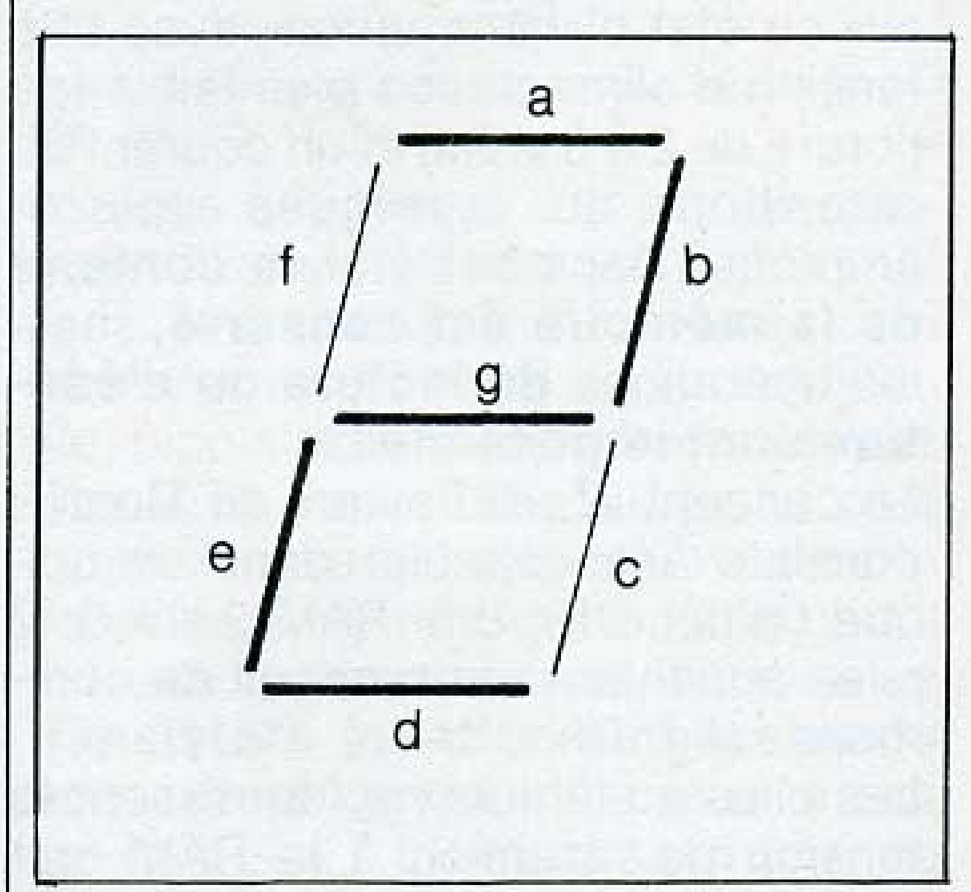


Fig. 84

Le tableau III donne l'équivalence entre le graphisme affiché et le code hexadécimal.

Avec cette représentation, on obtient 42 caractères. En réalité, il y en a 42 sans point décimal et autant avec le point décimal. La possibilité maxi-

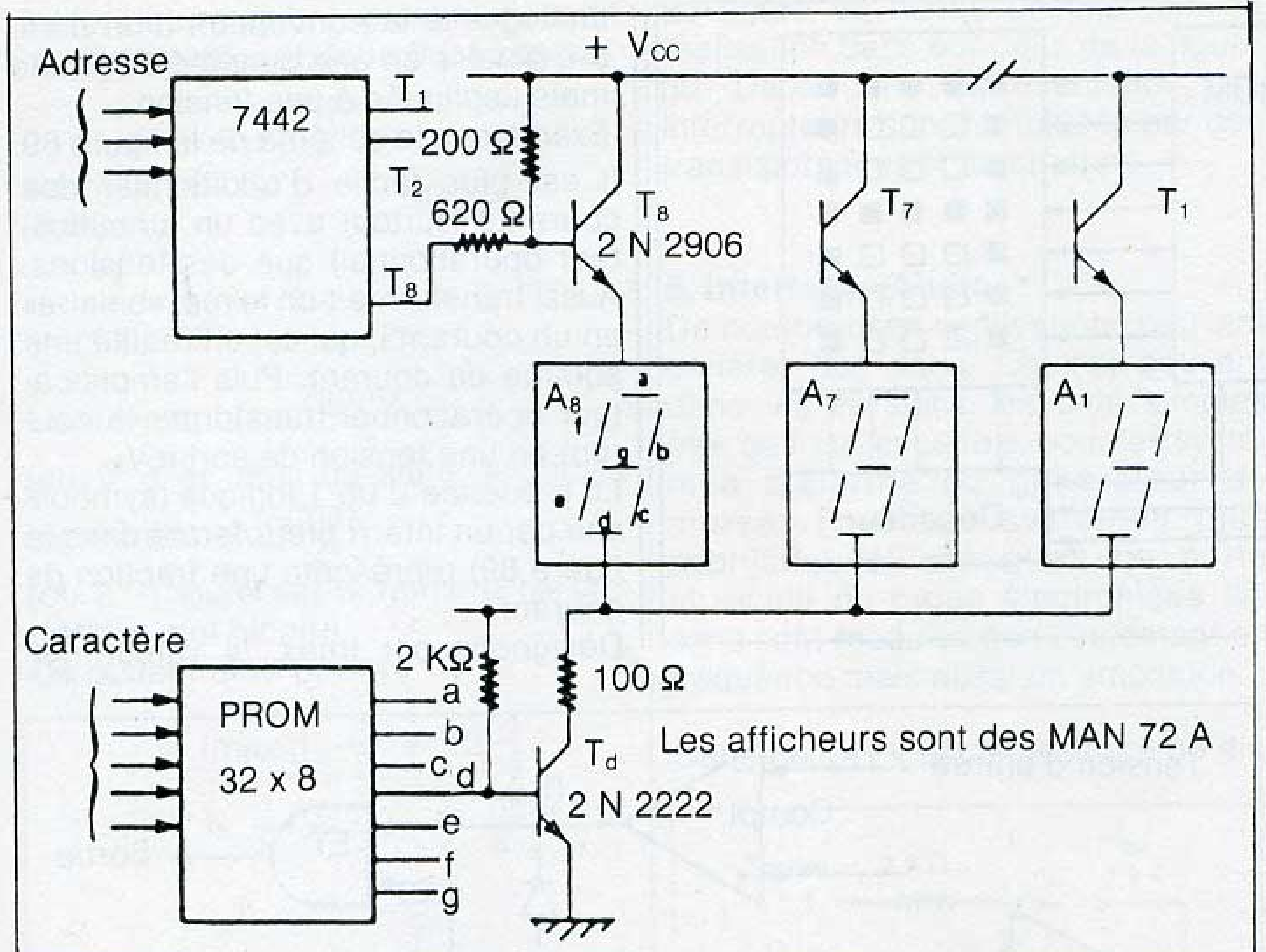


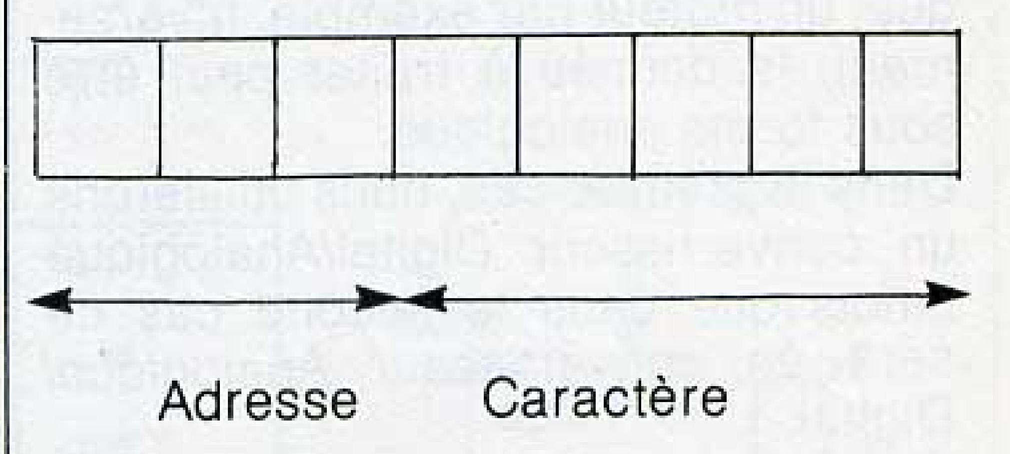
Fig. 85

Code	B D	30	9 B	B A	36	A E	A F	38	B F
Caractère	0	1	2	3	4	5	6	7	8
Graphisme									
Code	F E	3 F	A 7	8 D	B 3	8 F	0 F	A D	37
Caractère	9	A	B	C	D	E	F	G	H
Graphisme									
Code	89	B 1	97	B 5	2 B	23	A 3	1 F	3 E
Caractère	I	J	K	L	M	N	O	P	Q
Graphisme									
Code	0 3	A 6	87	B 5	B 7	A 9	0 7	B 6	8 A
Caractère	R	S	T	U	V	W	X	Y	Z
Graphisme									
Code	83	A 2	32	02	C 0	0 0			
Caractère	(	)	+	–		Blanc			
Graphisme									

Tableau III

male étant 2 fois 128 (2×2^7 combinaisons) mais certaines combinaisons ne présentent aucun intérêt.
b) La figure 85 représente un schéma complet d'un afficheur 8 digits.
L'octet contient d'une part le caractère à afficher (5 bits) et d'autre part son adresse (3 bits).
Examinons le schéma (figure 85).

Le décodeur 7442 décode l'un des huits afficheurs à sélectionner. Il reçoit sous forme de 3 digits l'«adresse» de l'afficheur. Par contre le PROM (ou générateur de caractères) qui en fonction du code caractère, sélectionne les segments à allumer.
La configuration de l'octet est dans ce cas la suivante :



3. Afficheur à matrice 5 x 7 :

Comme nous venons de le voir, le dispositif 7 segments permet d'approcher à peu près le graphisme des caractères. Pour obtenir une meilleure définition, l'emploi d'un système par points, par exemple matrice 5 x 7 s'impose (cinq colonnes et 7 points par colonne).
L'arrangement type d'un digit est donné par la figure 86.

Le principe du fonctionnement est le suivant : chaque caractère est décomposé en cinq colonnes, de chacune 7 points, soit au total 35 points.

La PROM reçoit les 8 bits du code ASCII, et pour chaque caractère génère 5 groupes de bits pour chacune des colonnes. Comme le montre la figure 86, il faut employer un système de balayage extérieur (horloge).

Dans la plupart des cas, la visu comporte 16, 20 ou 40 caractères, il faut donc adjoindre à ce système un dispositif de multiplexage.

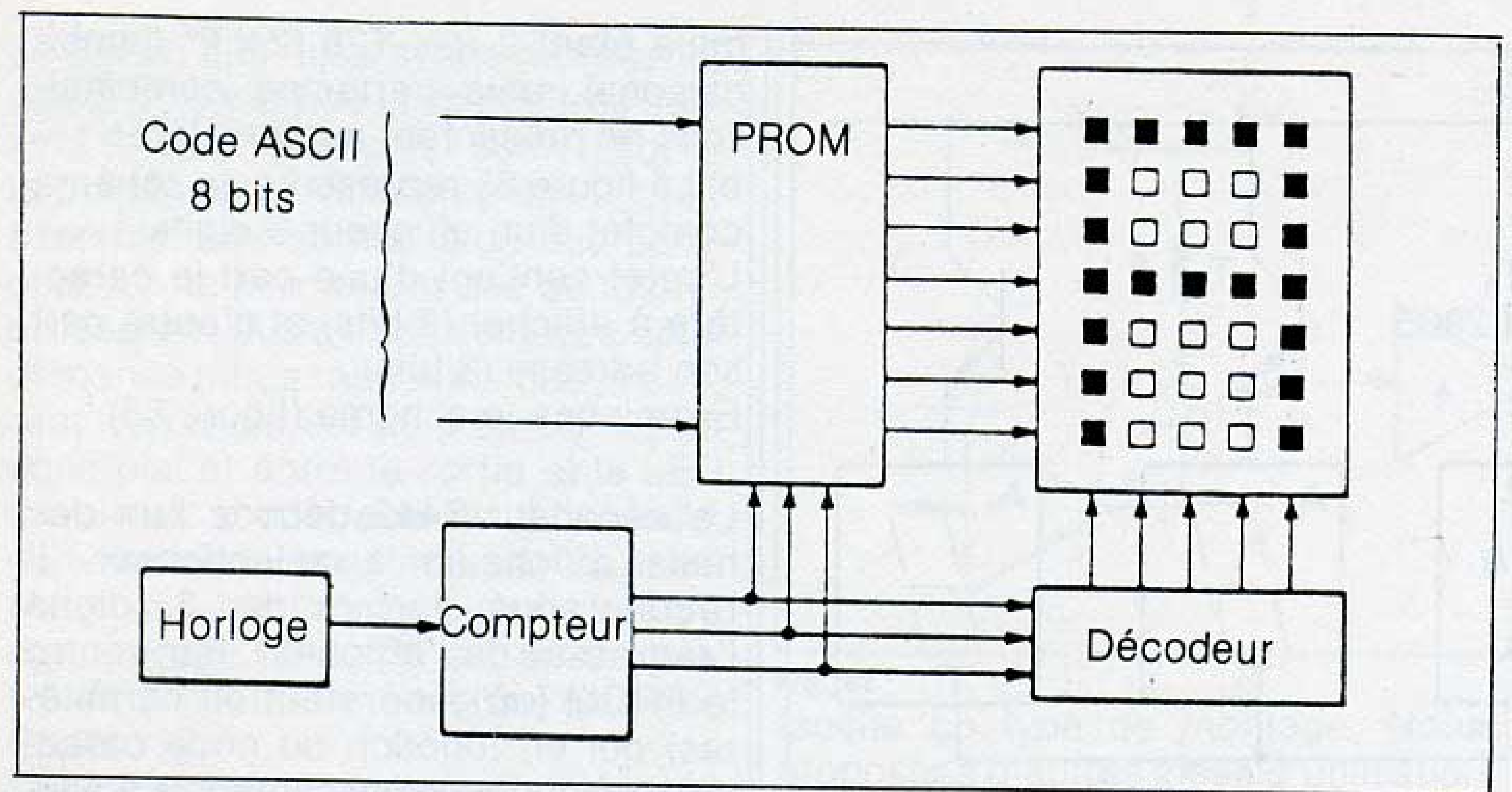


Fig. 86

analogue à la conversion d'un nombre binaire en une quantité décimale mais appliquée à une tension.
Examinons le schéma de la figure 89.
Il est plus facile d'additionner des courants (surtout avec un amplificateur opérationnel) que des tensions. Aussi transforme-t-on le mot «binaire» en un courant i, qui est en réalité une somme de courant. Puis l'amplificateur opérationnel transforme le courant en une tension de sortie V_S.
La présence d'un 1 logique (symbolisée par un interrupteur fermé dans la figure 89) représente une fraction de courant.
Désignons par Imax, la valeur $\frac{V_{ref}}{R}$

4. Les convertisseurs A/D et D/A :

Dans tout système microprocesseur, il faut souvent commander un organe périphérique par une tension analogique, un moteur par exemple. Inversement, la donnée à traiter peut être sous forme analogique.
Dans le premier cas, nous utiliserons un convertisseur Digital/Analogique tandis que dans le second cas ce sera un convertisseur Analogique/Digital.
Etudions brièvement ces deux circuits :

a) Convertisseur Analogique/Digital :

Le principe de fonctionnement est donné par le synoptique de la figure 87 et les chronogrammes de la figure 88.
La tension continue à convertir est appliquée sur l'entrée (−) d'un comparateur. Sur l'autre entrée (entrée +) on place une tension en rampe.
Tant que la tension de la dent de scie n'a pas atteint la tension à convertir, la porte laisse passer les impulsions (S_C à 1).
Dès que le comparateur détecte l'égalité, la porte se ferme (S_C à 0). Le nombre d'impulsions est proportionnel au temps qu'a mis la rampe pour atteindre la tension, donc proportionnel à sa valeur.
Il suffit de compter ces impulsions dans un compteur à sortie parallèle, de lui adjoindre une logique de démarrage et l'on obtient ainsi un convertisseur Analogique/Digital.

b) Convertisseur Digital/Analogique :

La conversion Digital/Analogique est

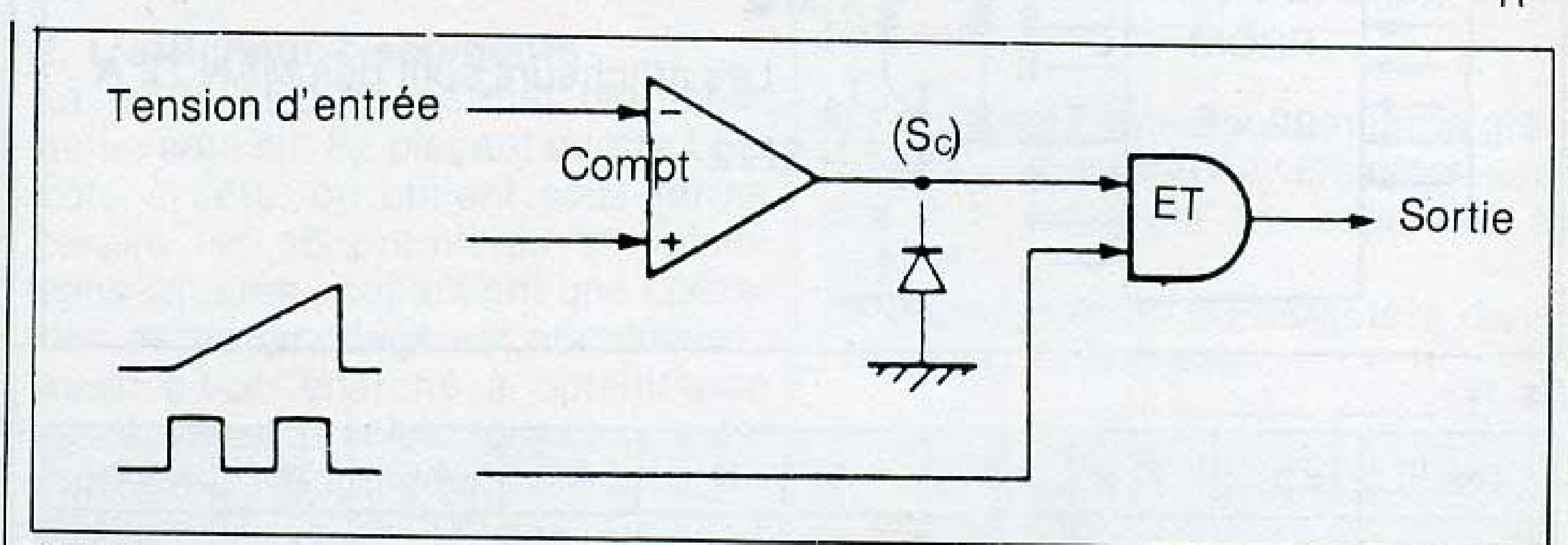


Fig. 87

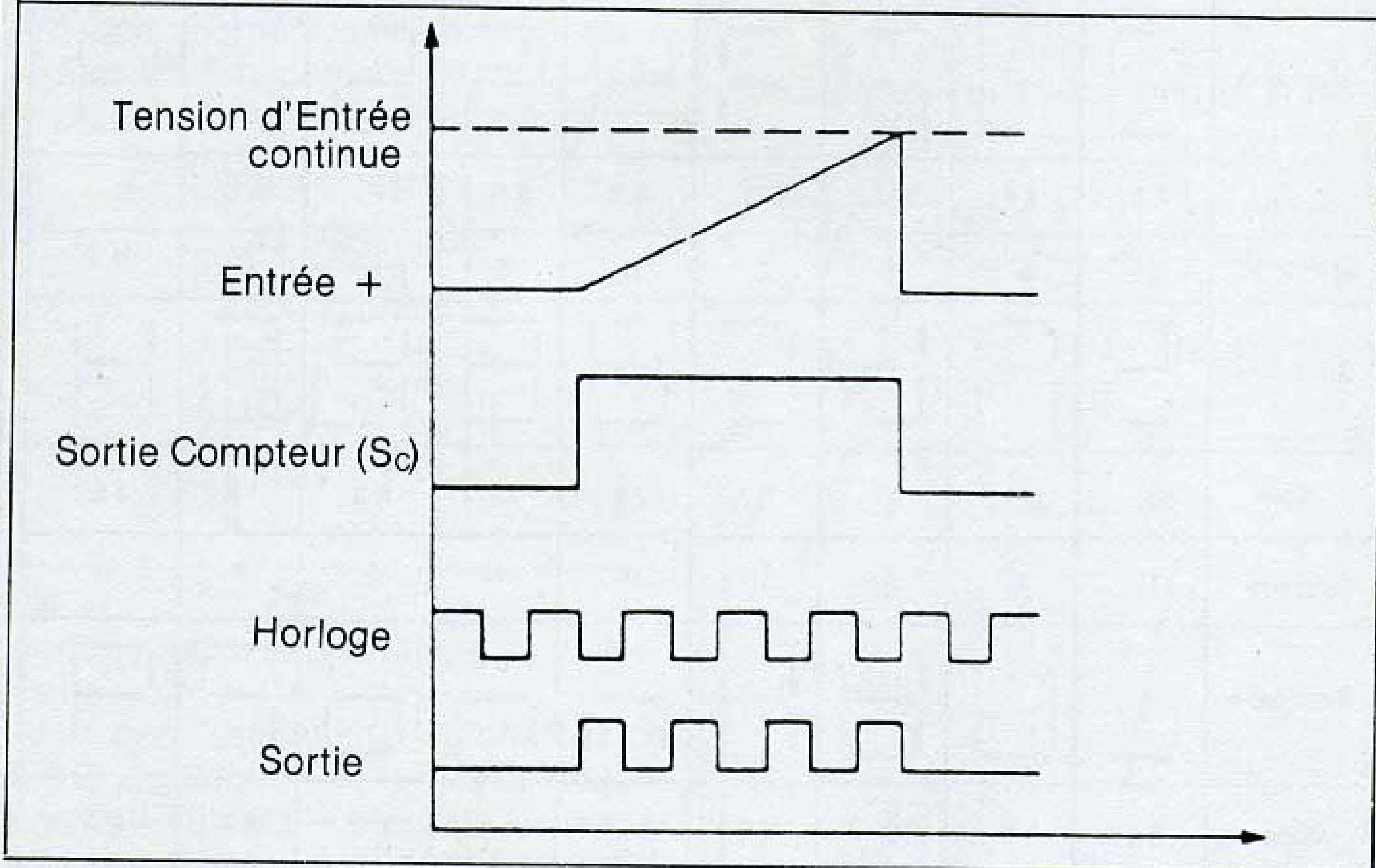


Fig. 88

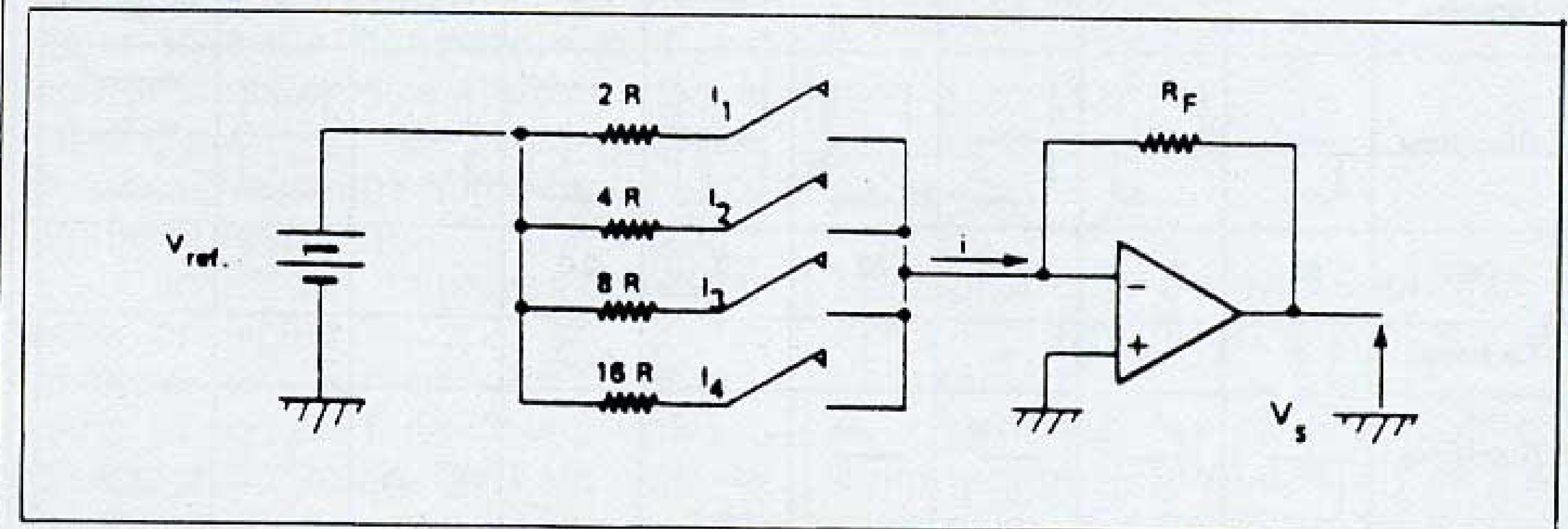


Fig. 89

La fermeture de I_1, représentant le bit le plus significatif, fournit un courant $\frac{Imax}{2}$

ou $Imax \times (2^{-1})$.

[Rappel

$$\frac{1}{2} = \frac{1}{2^1} = 2^{-1} ; \frac{1}{2^n} = 2^{-n}]$$

Le suivant, I_2 vaut $\frac{Imax}{4}$

(ou 2^{-2}) et ainsi de suite, le moins significatif vaut $\frac{Imax}{2^n}$

(ou 2^{-n}), ou n est le nombre de bits dans le mot binaire.

On obtient ainsi pour i :

$$= Imax\,(I_1.\frac{1}{2} + I2.\frac{1}{4} + I_3.\frac{1}{8} + ... + I_i.\frac{1}{2^i} + ... + I_n.\frac{1}{2^n})$$

de sorte que lorsque $I_1 = I_2 = I_3 = ... = I_i = ... = I_n = 1$

$i \simeq Imax$

Dans l'équation 1), les valeurs de I_1, I_2, I_3,... I_i,... I_n se confondent avec les valeurs logiques «0» ou «1» de chaque bit du mot binaire.

La tension V_S est telle que $V_S = -i.R_F$

$$\text{Lorsque } i = Imax = \frac{V_{ref}}{R}$$

$$(V_S)max = -\frac{V_{ref}}{R}.R_F \text{ si } R_F = R$$

$$Vmax = -V_{ref}$$

La figure 90 donne un exemple de réalisation basé sur celui de la figure 89. Dans cette représentation, les interrupteurs sont remplacés par des transistors à effet de champs.

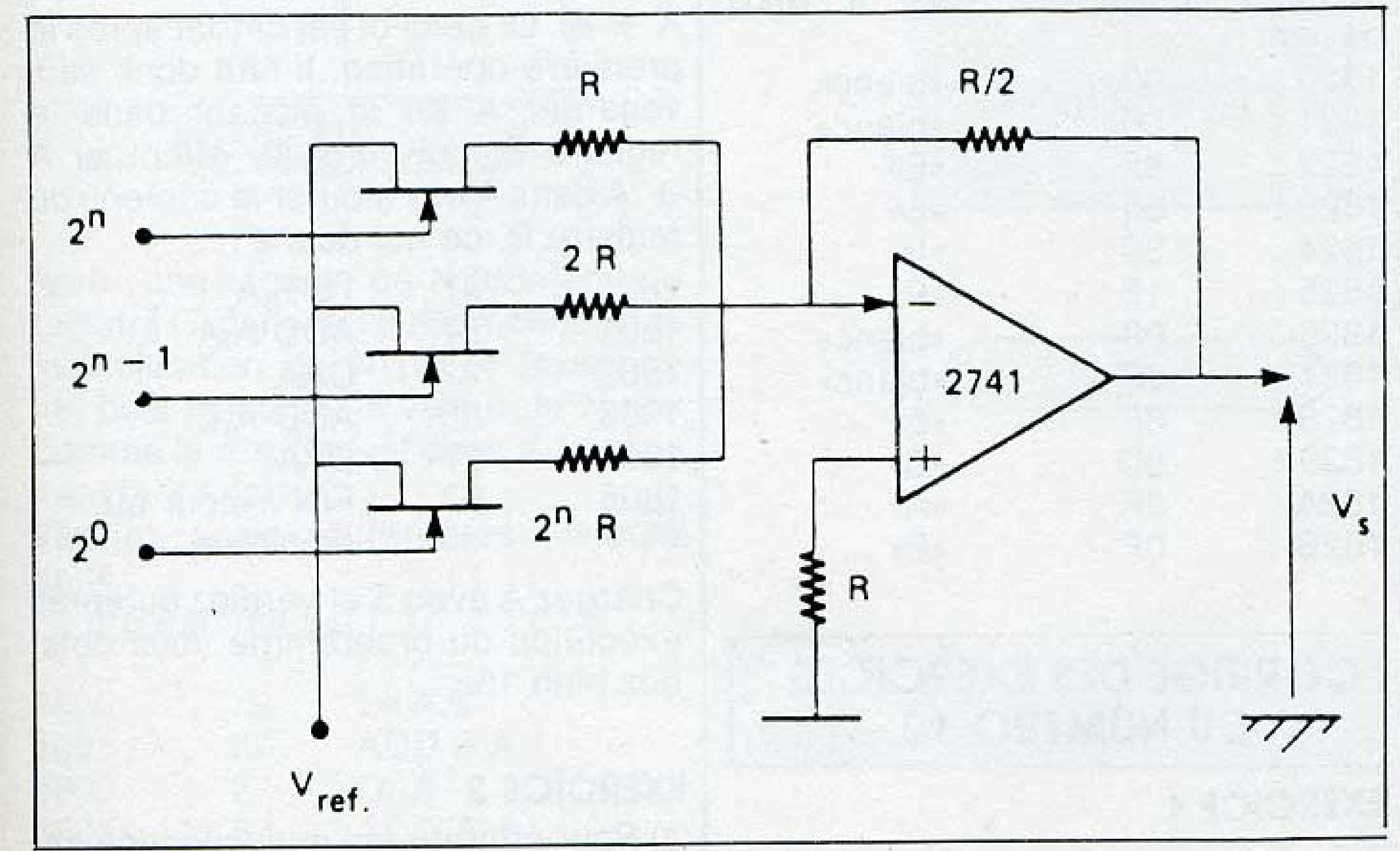


Fig. 90

5. Interface «Audio» :

De nombreuses applications peuvent générer des sons. Ceux-ci peuvent dans les versions les plus simples être de fréquence fixe, pour les systèmes d'alarme ou générateur de morse. Pour les systèmes plus sophistiqués, sonnettes de porte musicale ou orgue électronique les sons sont modulés non seulement en fréquence mais aussi en amplitude.

Dans ce cas il est nécessaire de disposer d'une interface «Audio» qui permette d'attaquer un haut-parleur. En utilisant plusieurs sorties du circuit périphérique et quelques résistances, on peut programmer le niveau sonore.

La figure 91 représente un amplificateur «audio» réalisé avec un circuit intégré LM 386 et quelques composants discrets.

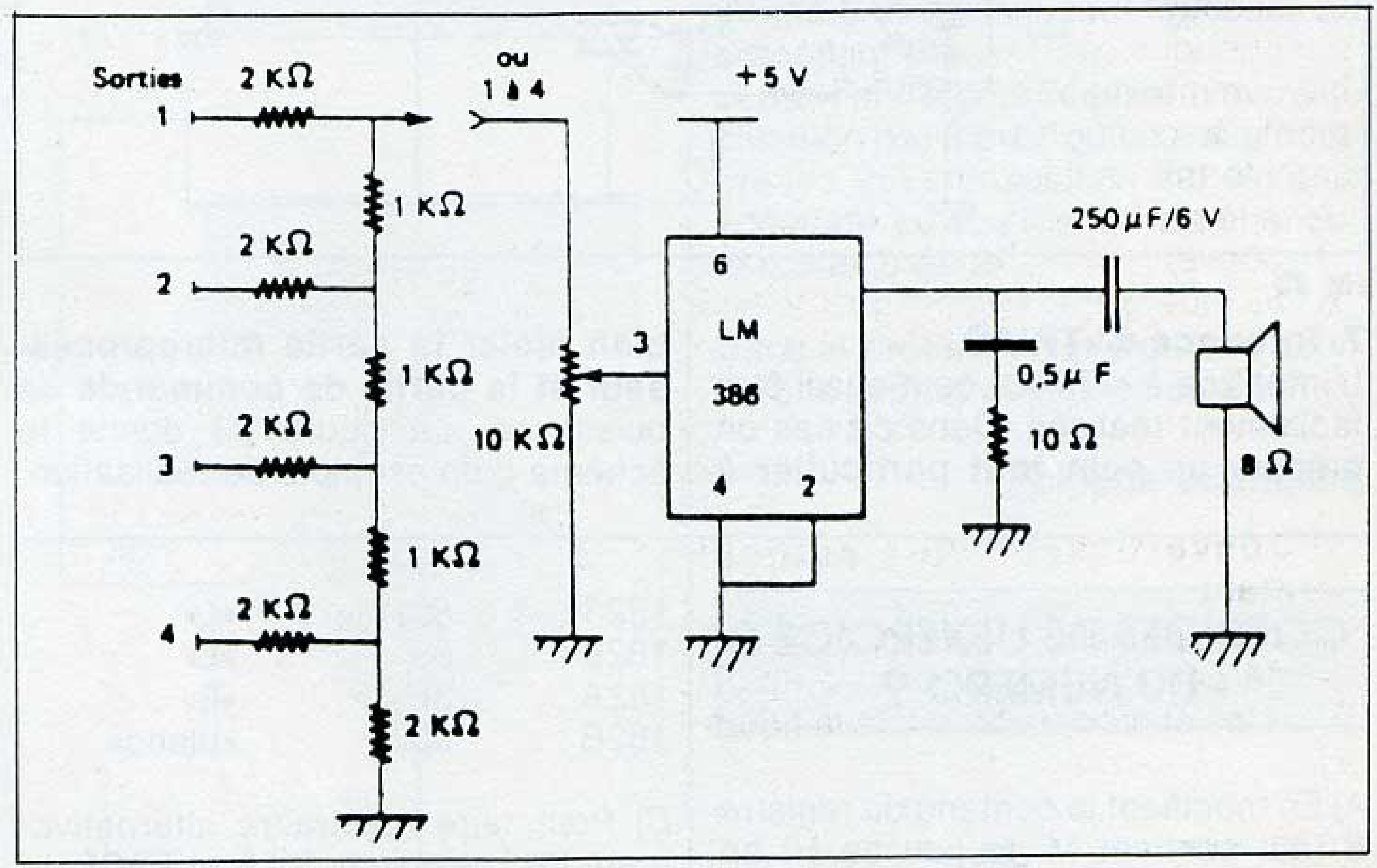


Fig. 91

6. Interface à relais :

Pour la commande d'une charge alimentée en alternative, une solution aisée est l'emploi d'un relais (ou deux quand la charge est importante) :

La figure 92 représente une méthode simple pour réaliser la commande d'un relais, lequel peut commander une charge plus conséquente.

Le relais utilisé a une bobine de 6 volts et ne consomme que 12 mA. Pour obtenir un courant suffisant, il faudra cependant connecter plusieurs sorties entre elles pour que la somme des courants soit suffisante afin d'actionner le relais.

Cette solution «économique» n'est pas recommandable car une sortie peut être amenée à délivrer un courant supérieur au maximum autorisé ; il vaut mieux utiliser un étage driver à transistor ou un circuit T.T.L.

La charge peut être un relais elle-même qui commande une charge plus importante : moteur, résistance chauffante, etc...

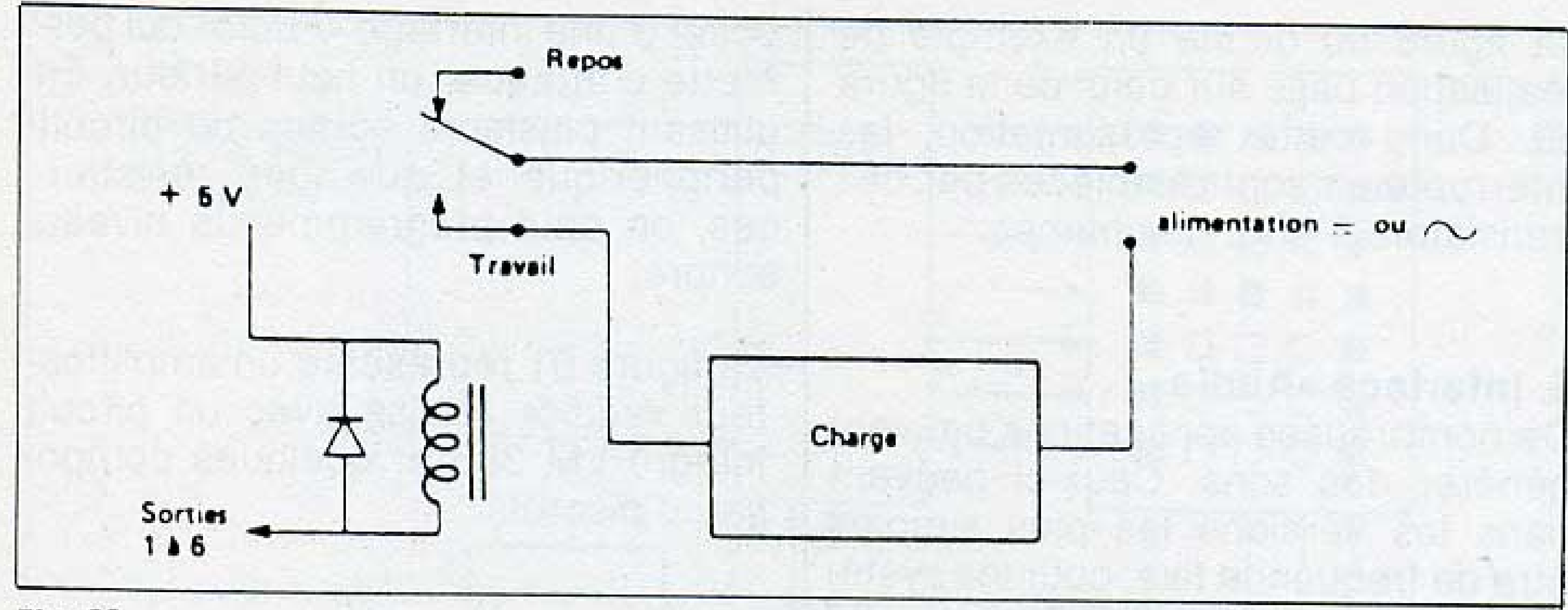


Fig. 92

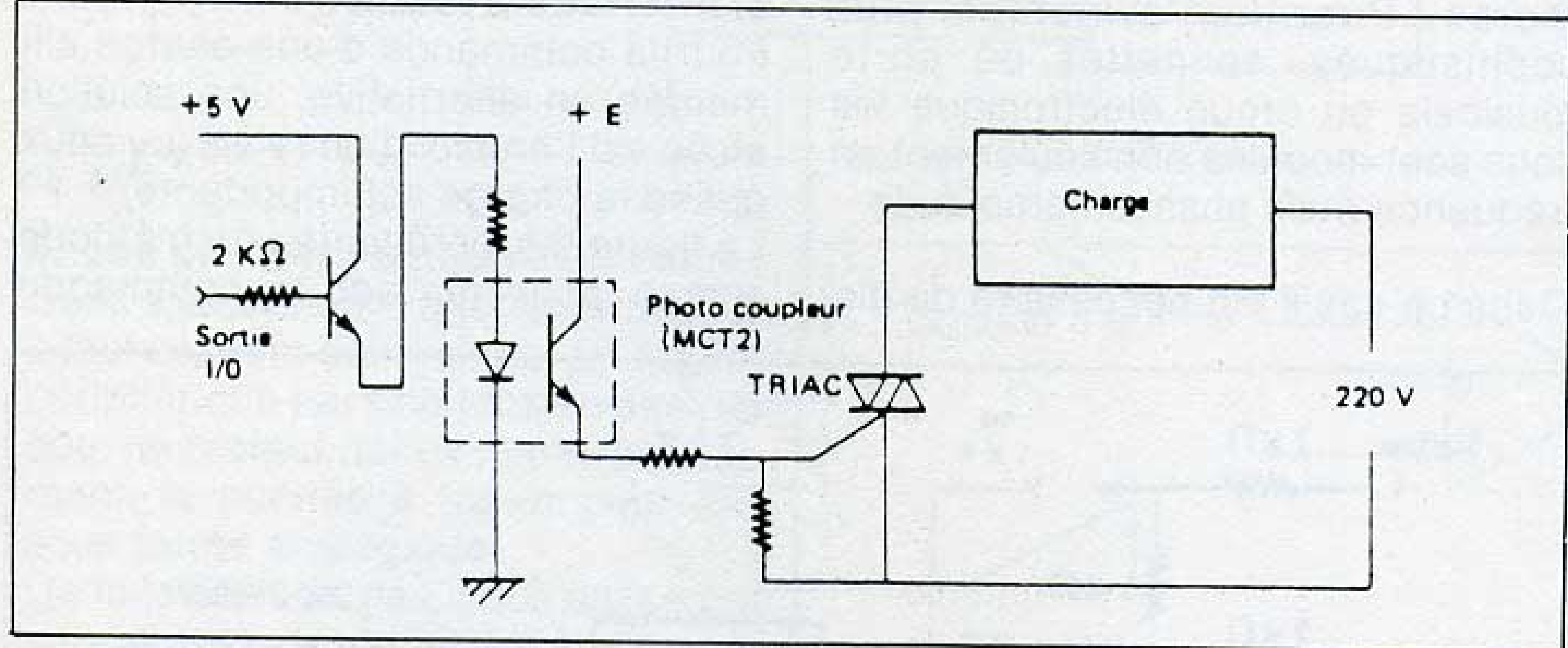


Fig. 93

7. Interface à «TRIAC» :
L'interface à «TRIAC» peut aussi être facilement réalisée. Dans ce cas on prendra un **soin tout particulier à bien isoler la partie microprocesseur et la partie de commande** de puissance. La figure 93 donne le schéma d'un exemple de réalisation.

CORRIGES DE L'EXERCICE 2 DU NUMERO 9

A) En modifiant le contenu du registre B (qui contient M au lieu de H) on constate que le clignotement s'accélère. Inversement quand le contenu de B augmente (64M au lieu de 32H) le phénomène se ralentit.
Le registre B est utilisé comme compteur. L'instruction suivante «CALL SCAM 1» qui affiche un message donné pendant une quinzaine de millisecondes sera répété autant de fois que l'indique le registre B. La diminution de 1 et le test sont effectués par l'instruction suivante DJNZ que nous avons décrite dans le N° 10, p. 62.
B) Pour faire clignoter la lettre H, il faut écrire le message «ELLO» au lieu des blancs dans la deuxième partie du message, ce qui donne :

1820 à	1825	inchangés
1826 :	00	«blanc»
1827 :	BD	«D»
1828 :	85	«L»
1829 :	85	«L»
182A :	8F	«E»
182B :	00	«blanc»

C) Pour faire apparaître alternativement les messages PILE et FACE, le contenu des cases 1820 à 182B devient :

1920 :	00	«blanc»
1821 :	00	«blanc»
1822 :	8F	«E»
1823 :	85	«L»
1824 :	30	«I»
1825 :	1F	«P»
1826 :	00	«blanc»
1827 :	00	«blanc»
1828 :	8F	«E»
1829 :	8D	«C»
182A :	3F	«A»
182B :	0F	«F»

CORRIGE DES EXERCICES DU NUMERO 10

EXERCICE 1

Le code 97 correspond à l'instruction A ← A - A ; c'est-à-dire soustraire de l'accumulateur la quantité qu'il contient et déposer dans A le résultat, ce qui donne 0. C'est une méthode de remise à zéro de A.
On obtient un résultat identique en chargeant A avec 00 c'est-à-dire

Ld A,00 ou 3E 00

mais dans ce dernier cas il faut utiliser 2 octets au lieu de 1.

EXERCICE 2

a) Réaliser 2 x A avec résultat dans A (code BCD).
Il suffit de faire A ← A + A, c'est-à-dire additionner au contenu de A, la valeur qu'il contient et de déposer le résultat dans l'accumulateur. Comme il s'agit de nombres exprimés en BCD, il faut ensuite réaliser un ajustement décimal (DAA) :

1800	87	ADD A,A
180	27	DAA
1802	F7	FIN (retour au moniteur)

b) Réaliser 4 x A avec résultat dans A (code BCD)

1800	87	ADD A,A
1801	27	DAA
1802	87	ADD A,A
1803	27	DAA
1804	F7	FIN (retour au moniteur)

c) Réaliser 3 x A avec résultat dans A (code BCD).
Il faut additionner à 2 fois A, une nouvelle fois le contenu de A (3 x A = 2 x A + A). Or celui-ci est détruit après la première opération. Il faut donc sauvegarder A en le plaçant dans le registre B, puis ensuite effectuer A + A dans A et y ajouter le contenu du registre B, ce qui donne :

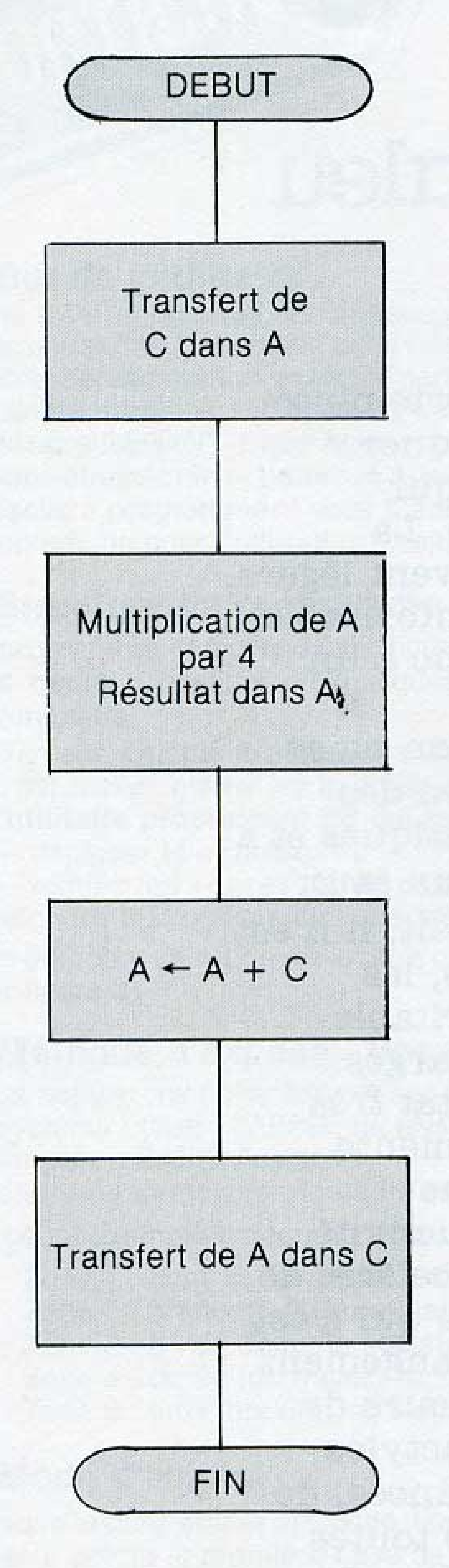
1800	47	Ld B,A
1801	87	ADD A,A
1802	27	DAA
1903	80	ADD A,B
1804	27	DAA
1805	F7	FIN (retour au moniteur)

Chargez A avec 5 et vérifiez qu'après exécution du programme vous obtenez bien 15.

EXERCICE 3

a) Pour effectuer la multiplication par 5 du contenu de C nous allons effec-

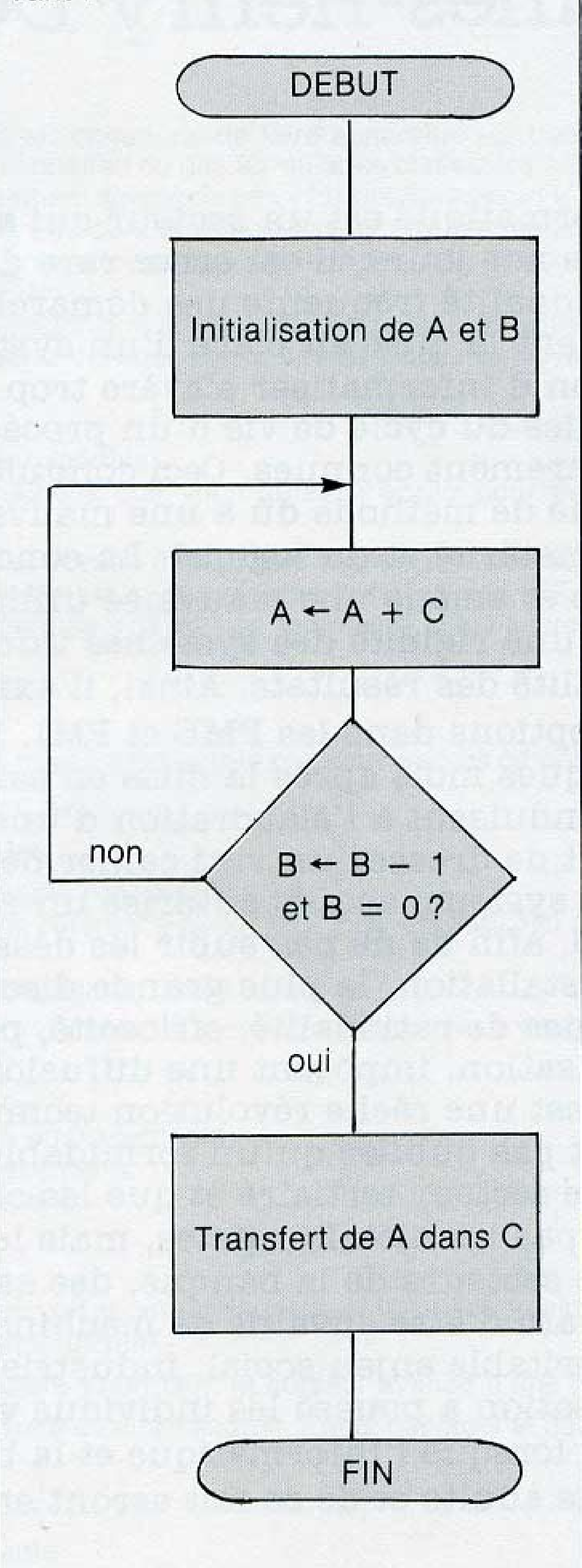
tuer un programme dérivé de l'exercice II.
Les différentes phases sont représentées par l'organigramme donné ci-après.



Après une recopie de N (contenu du registre C) dans A, vous procédez à la multiplication de N par 4 (exercice IIb) puis ajoutez-y la valeur de N. Comme le résultat est dans A, vous le transferez dans C.
Vérifiez avec différentes valeurs de N.
Le programme est :

```
1800    79      Ld A,C
1801    87      ADD A,A
1802    27      DAA
1803    87      ADD A,A
1804    27      DAA
1805    81      ADD A,C
1806    27      DAA
1807    4F      Ld C,A
1808    F7      FIN (retour au
                moniteur)
```

b) On obtient un résultat identique avec l'emploi de l'instruction DJNZ. Le nouvel organigramme est le suivant :



Après initialisation du registre A (remise à zéro) et du registre B utilisé en compteur, vous réalisez autant de fois que nécessaire l'opération

$$A \leftarrow A + C$$

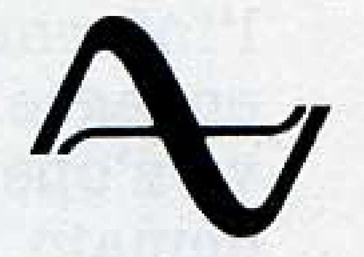
suivie d'un ajustement décimal. Le programme se termine par un transfert dans C.
Le programme est :

```
1800    97      Sub A,A
1801    06 05   Ld B,05
1803    87      ADD A,A
1804    27      DAA
1805    10 FC   DJNZ
1807    4F      Ld C,A
1808    F7      FIN (retour au
                moniteur)
```

Nous obtenons un second programme d'une longueur identique au précédent IIIa.
c) Par contre pour réaliser une multiplication par 3 ou 7 ou un autre nombre (< 255), la solution IIIb est plus adéquate puisqu'il suffit de changer le contenu de B.

Dans le prochain numéro nous aborderons la programmation du Z80[R].

Philippe Duquesne



COURS PRATIQUE DE MICROPROCESSEUR AVEC LE MICROPROFESSOR MPF-IB

CINQUIEME PARTIE

Le langage du Z80R (1)

SOMMAIRE

I. INTRODUCTION

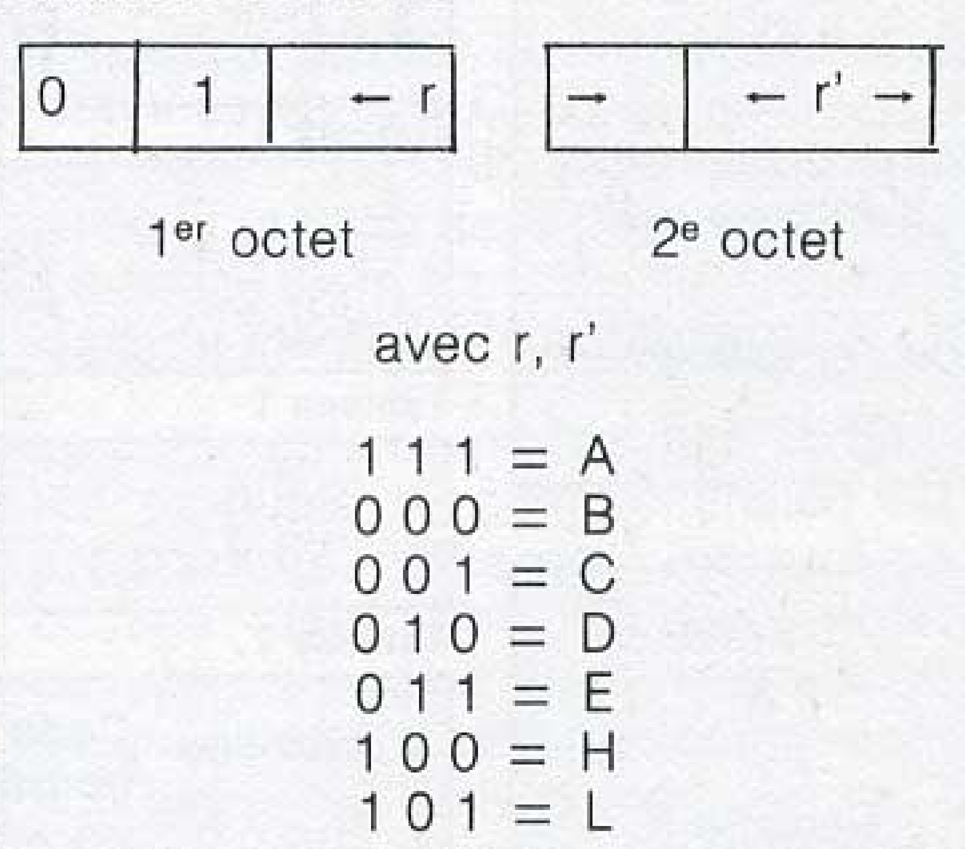
Les quatre premières parties de notre cours de microprocesseur avaient pour but de présenter au lecteur les différents éléments internes qui constituent un microprocesseur, ainsi que quelques composants externes (mémoires et circuits périphériques) indispensables.
Comme nous l'avons déjà dit, la connaissance des aspects matériels n'est pas réellement indispensable pour une bonne compréhension du logiciel, mais elle y contribue cependant beaucoup. Aussi n'hésitez pas à effectuer de temps à autre un petit retour en arrière et notamment à la deuxième partie (LED-MICRO n° 10) intitulée «Hardware du Z80», de manière à bien «mémoriser» (pour rafraîchir...) l'architecture interne du micro.
Avec cette cinquième partie, nous abordons la seconde phase consacrée à l'aspect logiciel (software en anglais). De nombreux exemples et exercices, exécutables avec notre Microprofessor MPF-IB, vous permettront d'acquérir une bonne maîtrise de la programmation en langage machine.
Nous présentons à la fin de cette partie, le répertoire complet du jeu d'instructions du Z80R. Pour chacune d'elles, nous indiquons le code mnémonique et un bref descriptif.
Les instructions sont rangées dans un ordre alphabétique, qui n'est pas celui de notre étude pour laquelle nous avons préféré un ordre par complexité croissante, mieux adapté.

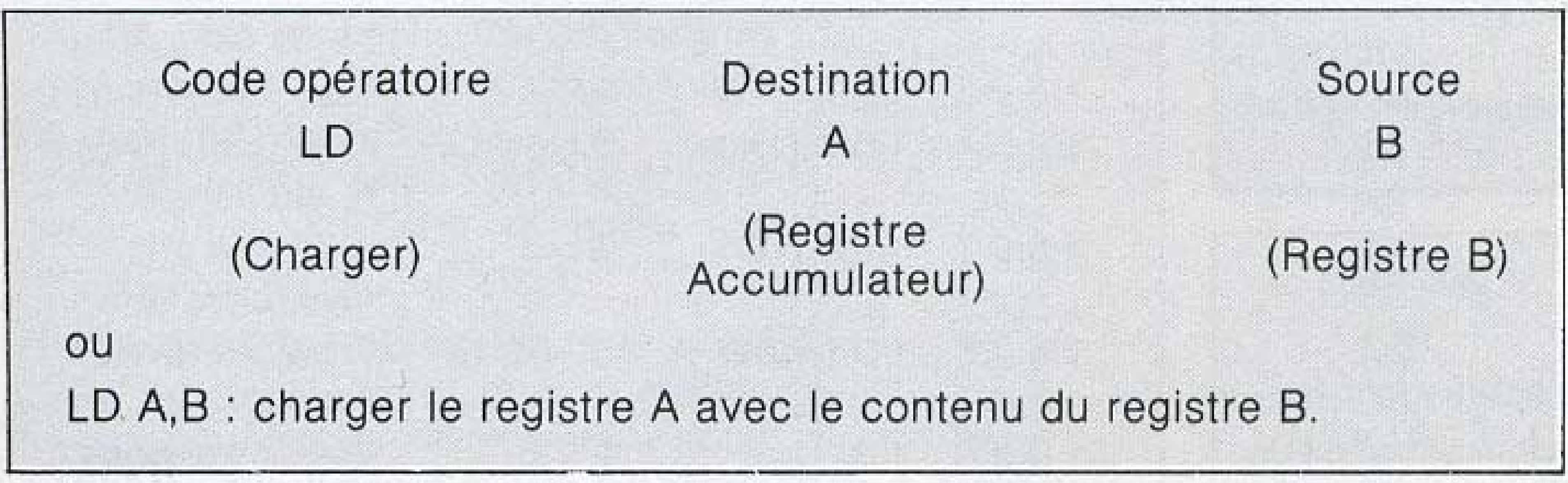
II. TRANSFERT DE DONNEES (1 OCTET)

II.1. Définition d'un TRANSFERT
Historiquement, on distinguait deux types de mouvement de données. Le premier permettait de charger (LOAD) un registre du CPU avec le contenu d'une case mémoire. Le second, l'opération inverse, consistait à enregistrer une donnée dans la mémoire, elle portait le nom de stockage (store).
Actuellement, l'instruction «transfert» désigne n'importe quel mouvement de données. La source et la destination sont définies dans la partie «opérande» de l'instruction. Le mnémonique du code de transfert (comme d'ailleurs la plupart des mnémoniques) varie d'un microprocesseur à un autre. Il en est ainsi dans le cas du Z80R. Ce code est LD (abréviation de LOAD), tandis que pour le 8080, le mnémonique est MOV (abréviation de MOVE).
Toute instruction commençant par «LD» correspond à un mouvement de données : transfert intégral du contenu d'un registre dans un emplacement mémoire spéficié, ou copie conforme du contenu d'un emplacement mémoire dans un registre donné. Dans tous les cas, le registre ou emplacement mémoire «origine» constitue la «source» et reste inchangé après l'exécution de l'instruction.

L'emplacement qui reçoit la donnée est le «destinataire». Avant l'exécution de l'opération de transfert, le registre destinataire contient une certaine donnée qui sera détruite par l'opération «LOAD», ou la nouvelle donnée, objet du transfert, se substitue au contenu présent avant l'exécution de l'opération.
Les tailles (1 ou 2 octets) du destinataire et de la source doivent être identiques. Nous reviendrons sur ce point très important dans les transferts de 2 octets.
Le schéma symbolique d'une opération de transfert est le schéma 1.

Et le code binaire s'obtient de la manière suivante :





Code opératoire	Destination	Source
LD	A	B
(Charger)	(Registre Accumulateur)	(Registre B)

ou

LD A,B : charger le registre A avec le contenu du registre B.

Schéma 1

Il est important de noter que la **«DESTINATION» précède la SOURCE** dans l'écriture symbolique.
Nous étudierons tout d'abord les «transferts de données» qui se présentent sous la forme d'un OCTET (8 bits).

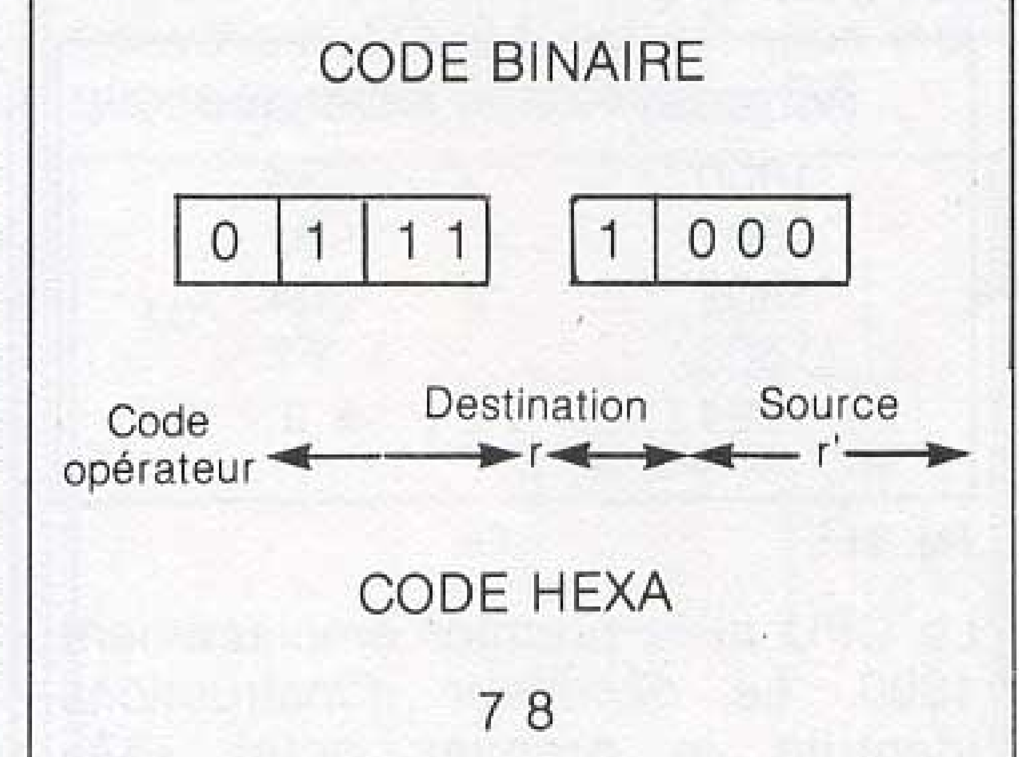
II.2. Transfert entre registres

Nous avons vu dans ce qui précède (LED-MICRO n° 10) que le CPU dispose, en plus du registre Accumulateur A, de six autres registres auxiliaires. Dans les opérations de transfert entre registres, chacun d'eux peut être soit la «source» soit la «DESTINATION». Le nombre d'instructions de transfert entre registres est de 7 × 7, soit 49.
Nota : Nous rappelons qu'à un instant donné, un seul jeu de registres A, B, C, D, E, H ou A', B', C', D', E', H' et L' est **en service** tandis que l'autre est en **«stand by» : ce qui exclut les transferts de registre à registre prime et vice versa.**
Le code symbolique des opérations de transfert entre registres est :

LD r', r

avec :
r : registre destinataire
r' : registre source.
(Les registres r et r' appartiennent **au même jeu de registres,** voir la note qui précède).

Exemple :
Etablissons le code de LD A,B.
Pour cela, il suffit de remplacer r par 111 qui représente le registre A et r' par 000 qui représente le registre B, ce qui donne :



Exercice 1
Avec le MPF-1B, charger respectivement sous le contrôle du MONITEUR, les registres A avec 00 et B avec 24H ([REG], [AF], 00, etc.)
Introduire le programme suivant à partir de 1800H :

1800	78	LD A, B
1801	F7	FIN (retour au moniteur)

a) Lire le contenu des registres A et B et vérifier qu'ils contiennent bien respectivement 00 et 24.
b) Exécuter le programme qui commence en 1800.
c) Lire les contenus des registres A et B.
Vous devez constater que le contenu du registre source B contient toujours 24 (inchangé) tandis que le registre destinataire A contient désormais lui aussi 24 au lieu de 00.

Exercice 2
Pour vous entraîner, déterminez les codes hexadécimaux pour les instructions suivantes :
LD C, A
LD E, C
LD D, E
LD L, D
LD H, L
Constituez un second programme (à partir de 1802) avec les cinq instructions de chargement, dans l'ordre indiqué, sans oublier de le terminer par F7. Introduisez-le et exécutez-le. Examinez tous les registres, ils doivent tous contenir 24.

Réponse :

1802	4F	LD C, A
1803	59	LD E, C
1804	53	LD D, E
1805	6A	LD L, D
1806	65	LD H, L
1807	F7	FIN. Retour au moniteur

II.3. Transferts entre registre A et case mémoire.

Dans le paragraphe précédent, l'opération consistait à recopier le contenu d'un registre dans un autre registre : l'opération était interne au CPU. Nous étudierons maintenant les opérations de **transferts entre l'accumulateur et la mémoire externe.**
Dans ce type d'instruction, seul l'accumulateur peut intervenir comme registre «Destinataire» ou «Source».

Le schéma de l'instruction du chargement de A avec le contenu (1 octet) d'un emplacement mémoire quelconque désigné par «nn» (2 octets pour l'adresse)(voir schéma 2)

Exemple :
Soit à transférer dans A le contenu de l'emplacement d'adresse 04 B1. Qui, dans le cas du MPF-1B, appartient à la zone «MONITEUR» et contient 85 (voir page 29 du listing source).
Le programme correspondant à cette opération est indiqué au schéma 3.

Code opératoire	Destination	Source
LD	A	(nn)
Charger	Registre A	Contenu de l'emplacement d'adresse «nn»

ou

LD A,(nn) : charger A avec le contenu de l'emplacement d'adresse nn.

Schéma 2

Code opératoire	Destination	Source
LD	(nn)	A
Charger	(emplacement d'adresse nn)	Registre A

ou

LD (nn), A : charger l'emplacement d'adresse nn avec l'octet contenu dans le registre A.

Schéma 4

Code opératoire	Destination	Donnée
LD	r	n
(charger)	(registre r) ,	(l'octet «donnée»)

ou LD r,n : charger le registre r avec «n», où r qui désigne l'un des 7 registres comme dans l'instruction LD r,r' (figure 98).

Schéma 5

1 8 0 0	3A B1 04	LD A,(04 B1)
1 8 0 3	F7	FIN (retour au moniteur)

Schéma 3

1 8 0 0	32 00 19	LD (1900), A
1 8 0 3	F7	FIN (retour au moniteur)

Tableau 1

3 A	0011 **1**010	LD A,(nn)
3 2	0011 **0**010	LD (nn), A

Tableau 2

Adresse	Code opératoire	Code machine	Commentaires
1 8 0 0	LD A,(1A00)	3A 00 1A	Charge du 1er opérande dans A
1 8 0 3	LD C ← A	4F	Transfert A dans C
1 8 0 4	LD A,(1A01)	3A 01 1A	Charge du 2e opérande dans A
1 8 0 7	ADD A,C	81	Add de A à C, résultat dans A
1 8 0 8	LD (1A02), A	3E 02 1A	Transfert de A dans (nn)
1 8 0 B	FIN	F7	Retour au moniteur

Tableau 3

Après l'exécution de ce programme, le contenu du registre A est 85 : copie de l'emplacement 04 B1.

Quelques remarques s'imposent à propos de cette instruction LD A,(nn).

1. Seul le registre A peut être chargé à partir d'un emplacement mémoire quelconque. Le registre Accumulateur A est contenu **implicitement** dans le code opérateur 3A de l'instruction ; ce qui n'était pas le cas dans l'instruction LD r,r'.

2. L'adresse nn de la **source est placée entre parenthèses.** La présence des parenthèses ( ) signifie que le code qu'elles renferment ne doit pas être considéré comme une quantité mais comme l'**ADRESSE de l'EMPLACEMENT** qui contient la donnée sur laquelle porte l'opération (ici le transfert).

3. Le code objet correspondant à l'instruction LD A, (04 B1) est 3A B1 04. **L'octet de poids faible** B1, partie **inférieure de l'adresse précède l'octet de poids fort** 04. Cette inversion devra toujours avoir lieu pour les codes de deux octets qui représentent une adresse ou une quantité hexadécimale sur 16 bits.

Jusqu'à présent, nous n'avons rencontré que des instructions d'un seul octet, tandis que celle-ci en comporte 3.

Etudions ce qui se passe.

Le programme, défini plus haut, est chargé dans la mémoire RAM à partir de l'adresse 1800 H comme l'indique la figure 94.

Adresse	Donnée
1800	3A
1801	B1
1802	04
1803	F7
1804	..

Fig. 94

Le CPU lit le premier emplacement 1800. Le décodeur d'instructions identifie le premier octet «3A» comme étant le premier byte d'une instruction qui en comporte trois. **Le CPU procède alors, avant toute exécution, à la recherche des deux autres octets,** qu'il sait contigüs au premier. Comme ceux-ci sont respectivement les 8 bits de poids faible et les 8 bits de poids forts de l'adresse, ils sont déposés sur le bus d'adresses pour séléctionner l'emplacement «Source».

Puis l'exécution de transfert est exécutée.

L'opération inverse qui consiste à transférer le contenu du registre A dans un emplacement mémoire, présente une structure similaire. Dans ce cas, le registre A constitue la «source» tandis que l'emplacement d'adresses «nn» devient la «destination». A noter toutefois que l'emplacement doit appartenir à la zone mémoire vive, sinon l'opération serait inefficace. Dans l'instruction précédente, l'emplacement mémoire, qui constituait la source, pouvait être soit une zone mémoire morte, comme l'exemple l'indiquait, soit une zone mémoire vive, comme nous le montrerons dans un prochain exemple (voir schéma 4).

Le schéma est :

Exemple

Supposons que le registre A contienne l'octet 85 et que l'on souhaite le transférer dans la case d'adresse 1900. Le programme correspondant est :

Les trois remarques que nous avons faites pour l'instruction LD A,(nn) restent valables ; seul le contenu du registre A peut être transféré en mémoire, l'adresse de l'emplacement destinataire (exclusivement RAM) est placé entre parenthèses et comme précédemment le code opératoire 32 est suivi immédiatement de l'octet de poids faible (00) puis de l'octet de poids fort 19 de l'adresse.

L'examen **des codes binaires** des

deux instructions montre que seule la valeur du bit 3 du code opération diffère d'une instruction à l'autre (voir tableau 2).

Ces deux instructions nous permettent d'utiliser la mémoire vive comme une zone de stockage «longue durée», très étendue, et ainsi de libérer les registres auxiliaires.

II.4. Exemple

Nous en savons maintenant assez pour écrire un programme plus conséquent.
Objet : Réaliser l'addition du contenu des cases mémoires 1A00H et 1A01H, puis placer le résultat somme dans l'emplacement 1A02H. (Il s'agit de quantités décimales).
Ecrivez ce programme, introduisez-le dans votre MPF-1B à partir de l'adresse 1800H. Vérifiez.

Nota : Les quantités placées dans 1900 et 1901 sont des quantités hexadécimales, la somme sera donc effectuée en base 16. Pour obtenir une addition décimale, il suffit de rajouter DAA (code 27) après l'addition et de choisir des quantités initiales telles que leur somme soit inférieure à 99.

Essayez d'écrire le programme avant de lire la solution ci-après.

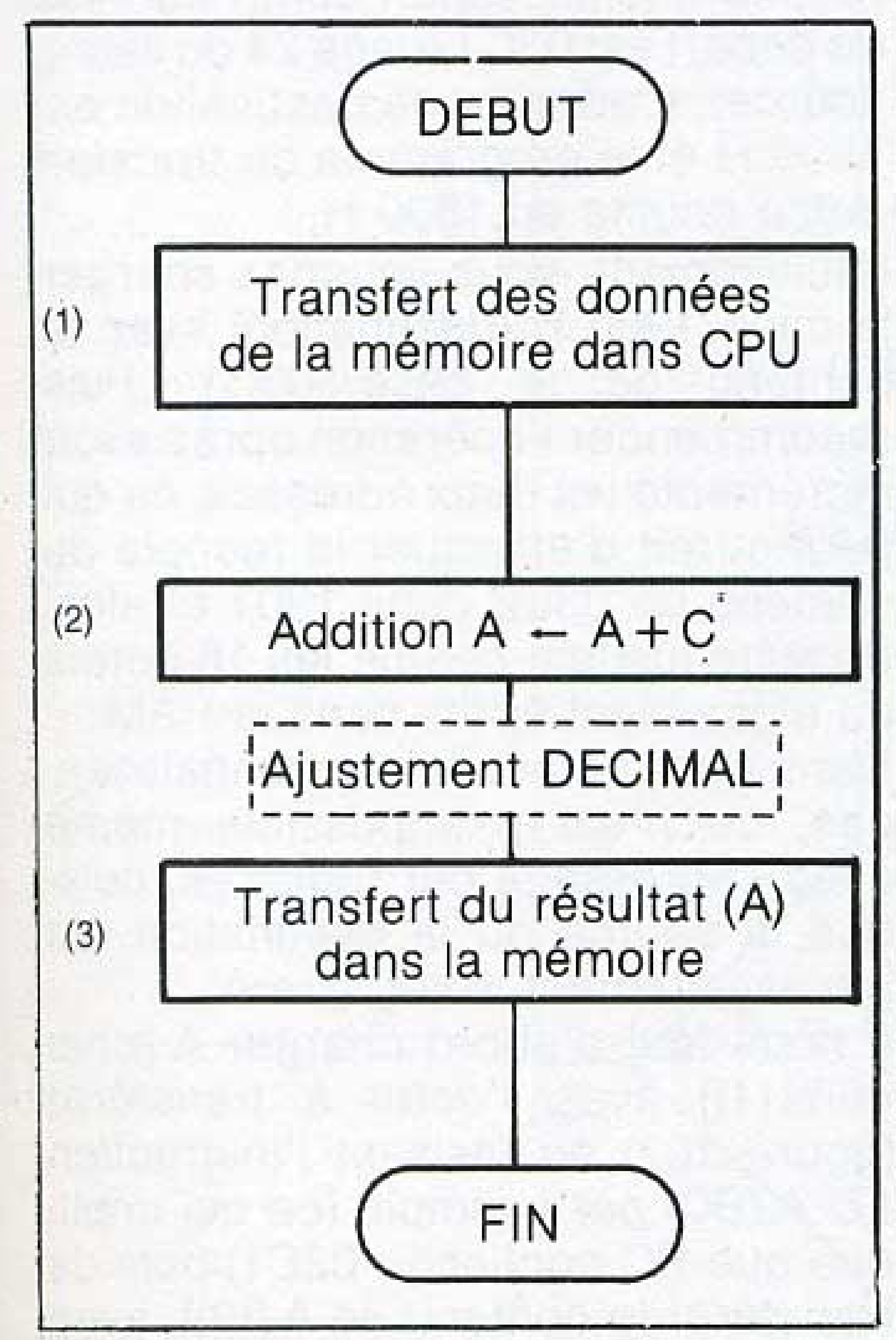


Fig. 95

SOLUTION

La figure 95 décrit d'une manière succincte les trois phases de notre programme. C'est un organigramme (ou Flow Chart).
La phase (1) est la préparation des données dans le CPU.
La phase (2) l'addition elle-même avec le résultat dans l'accumulateur.
La phase (3) le transfert de la somme dans la mémoire (voir tableau 3).

Le commentaire est facultatif. Il facilite toutefois la compréhension du programme. Chaque instruction occupe une ligne. Chaque octet occupe un emplacement «mémoire», ce qui conduit à un empilement continu des bytes dans la mémoire.
La figure 96 indique la disposition du programme dans la mémoire.

1800	3A
1801	00
1802	1A
1803	4F
1804	3A
1805	01
1806	1A
1807	81
1808	3E
1809	02
180A	1A
180B	F7

Fig. 96

Nota : Pour ceux qui n'auraient pas trouvé que LD C,A est 4F nous indiquons le détail ci-après (fig. 97) :

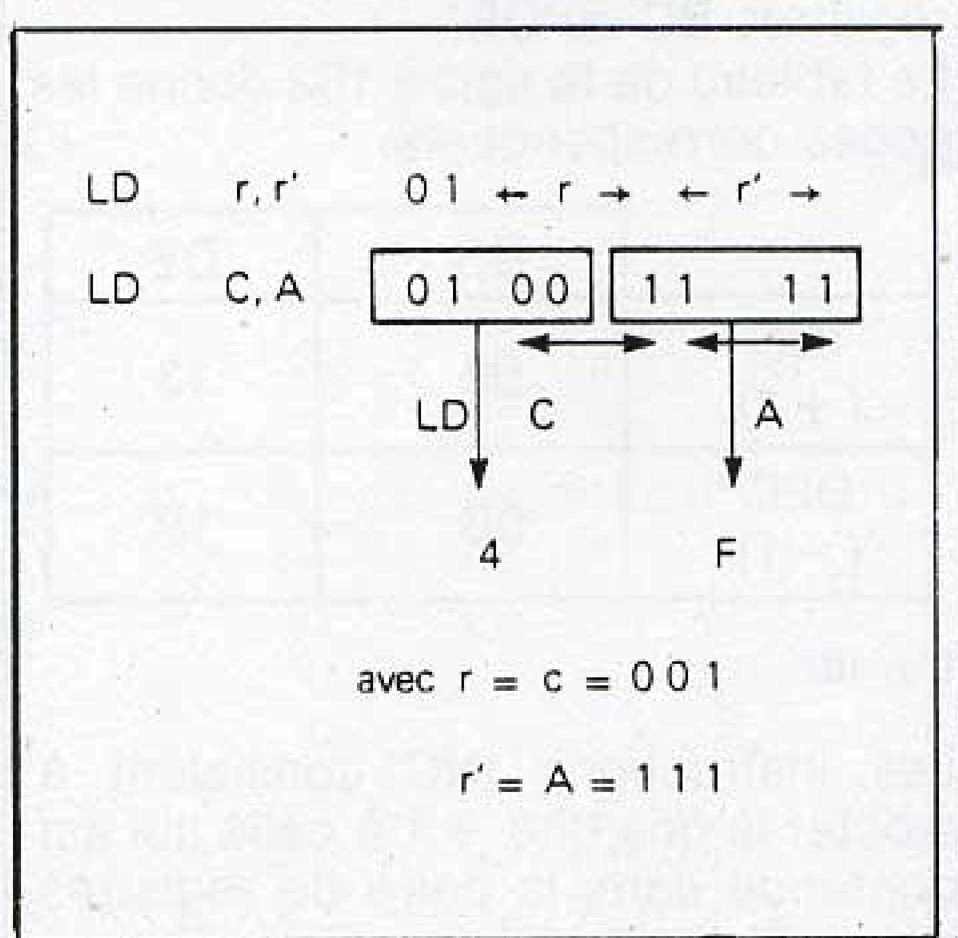


Fig. 97

II.5. Chargement immédiat

Une autre manière de charger un registre est d'utiliser les instructions de chargement immédiat. Le registre destinataire est identifié dans le premier octet et **la donnée est l'octet immédiatement suivant.**
Le schéma de l'instruction est donné au schéma 5.

Le code binaire est :

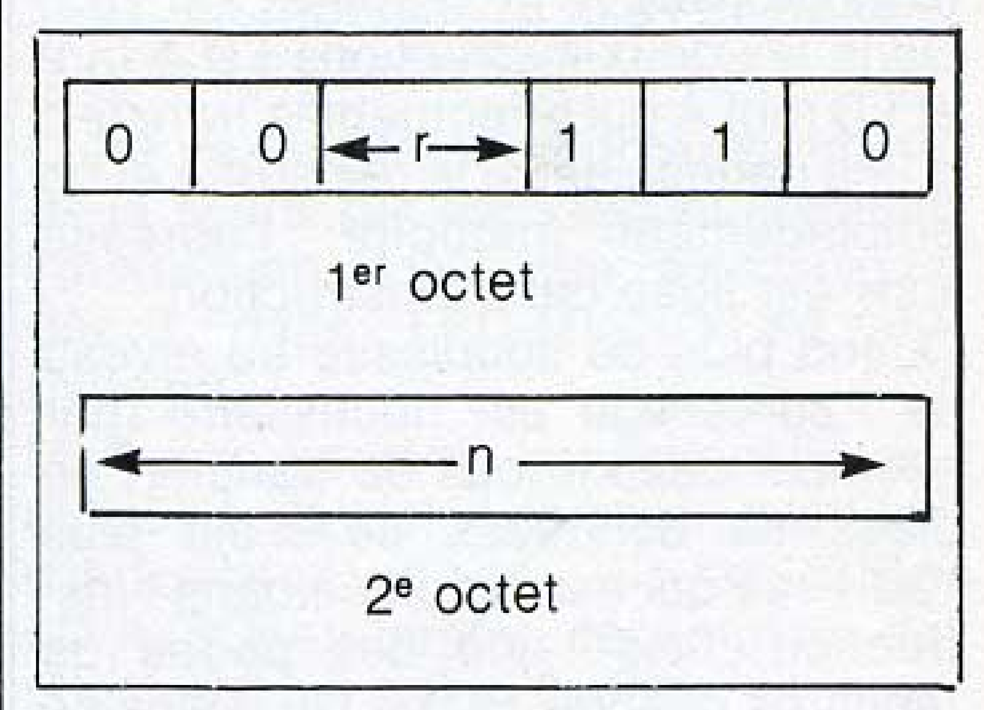


Fig. 98

Exemples
n représente une quantité hexadécimale comprise entre 00H et FFH, c'est-à-dire une valeur décimale arithmétique comprise entre 0 et 255.

1) Soit à charger le registre A avec 3F
Le code symbolique est LD A,3F ce qui donne pour le code binaire :

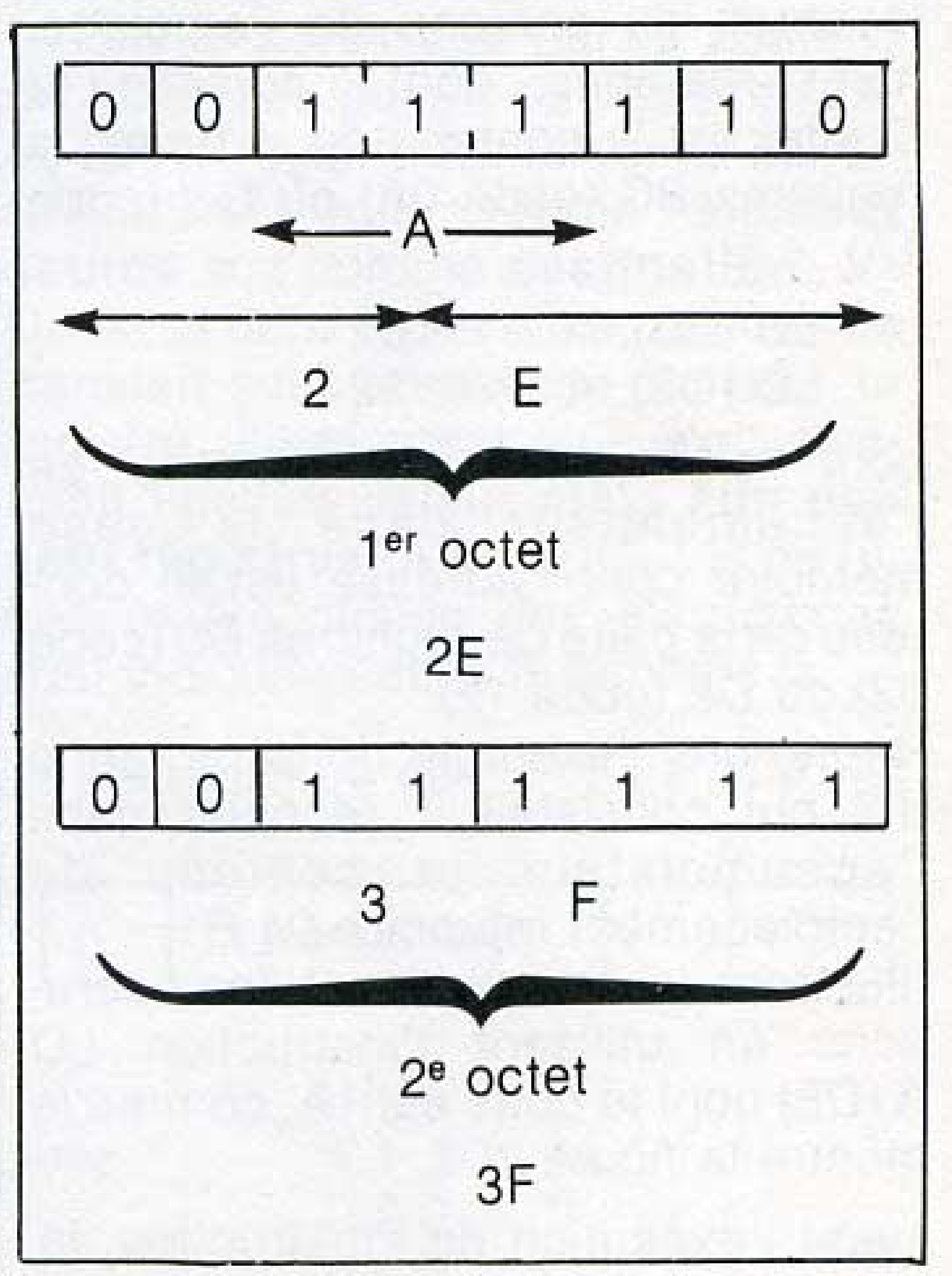


Fig. 99

et pour le code hexadécimal :
LD A,3F : 2E 3F

2) Soit à charger le registre B avec 76.
Le code binaire du premier octet est :

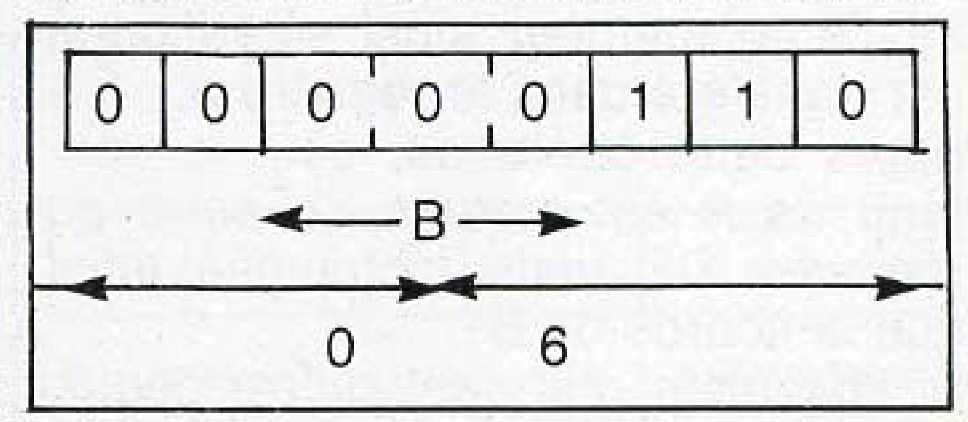


Fig. 100

ce qui donne pour le code hexadécimal : LD B,76 : 06 76

II.6. Transfert avec adressage par (BC) ou (DE)

Dans les deux instructions LD A,(nn) et LD (nn),A qui effectuent le transfert d'une donnée entre le registre A et un emplacement mémoire, l'adresse «nn» est figée dans l'instruction.

Quand plus de souplesse au niveau de l'adressage est nécessaire (par exemple dans le cas de transfert de blocs de données), ce n'est plus l'adresse qui est spécifiée dans l'instruction mais l'une des paires de registres doubles BC ou DE. Dans ce cas, **c'est le contenu de la paire de registres désignée qui constitue l'adresse «Source» ou «Destination».**

Les quatre codes mnémoniques et hexadécimaux sont :

a) LD A,(BC) 0A
LD A,(DE) 1A

pour le transfert dans l'accumulateur (1 octet) du contenu de l'emplacement mémoire dont l'adresse (2 octets) est le contenu de la paire de registres BC (code 0A) ou DE (code 1A).

b) LD (BC), A 02
LD (DE), A 12

pour le transfert du contenu de l'accumulateur dans la case mémoire dont l'adresse est le contenu de la paire de registres BC (code 02) ou DE (code 12).

Reprenons l'exemple du paragraphe II.3 qui consistait à recopier dans l'accumulateur le contenu de l'emplacement mémoire 04 B1.

Illustrons le cheminement des opérations en utilisant l'instruction LD A,(DE) dont le code est 1A, comme le montre la figure 101.

Avant l'exécution de l'instruction, la paire de registres DE contient la quantité 04 B1 (l'octet de poids faible dans E, l'octet de poids fort dans D).

Au moment de l'exécution de l'instruction 1A, le CPU utilise le contenu de DE comme pointeur, dépose celui-ci sur le bus d'adresses et le contenu de l'emplacement ainsi sélectionné est transféré dans le registre A.

Après cette opération, seul le contenu de A est modifié, la paire de registres DE reste inchangée ainsi que la source 04 B1.

Si l'exemple qui vient d'être décrit, détaille le fonctionnement des opérations de transfert avec adressage par registre, il n'en montre que partiellement l'intérêt.

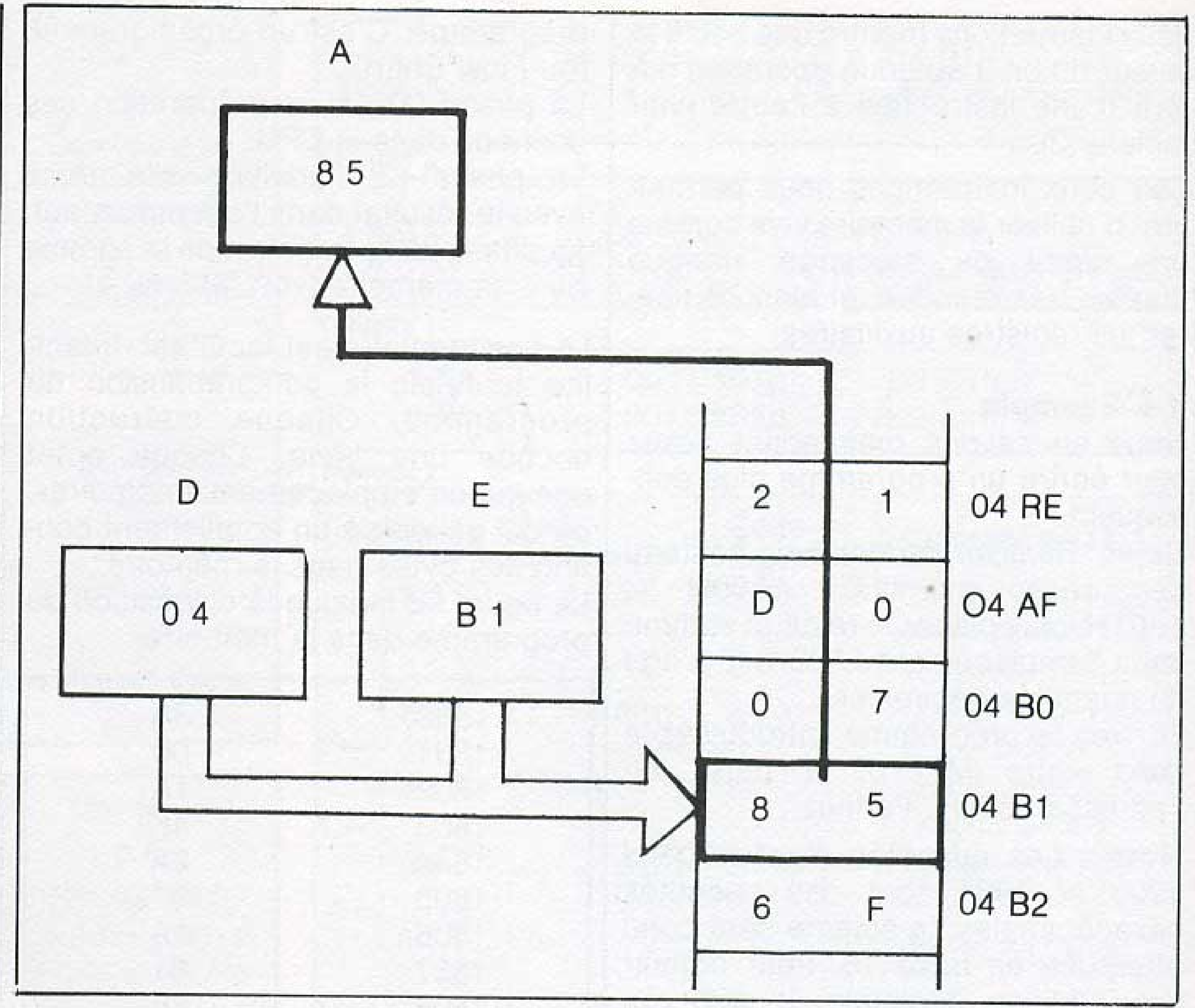


Fig. 101

Pour que le lecteur comprenne bien, nous allons donner un second exemple.

Auparavant, il nous faut anticiper sur le cours et introduire les instructions INC (incrémentation) et DEC (décrémentation) du contenu de la paire de registres BC ou DE.

Le tableau de la figure 102 donne les codes correspondants.

	BC	DE
INC (+1)	03	13
DEC (−1)	0B	1B

Fig. 102

Les instructions INC consistent à ajouter la quantité +1 à celle qui est contenue dans la paire de registres désignée.

Ainsi, si le contenu de DE est 2050 après l'exécution de l'instruction 13, le contenu est 2051.

De même, si BC contient 1EFF après l'exécution de l'instruction 03, le contenu est 1F00.

Les instructions DEC opérent d'une manière identique, mais la quantité 1 est retirée du contenu de la paire de registres au lieu d'y être ajoutée.

Le problème à traiter consiste à transférer un bloc de 18 (décimal) octets de la zone mémoire ROM pour le placer dans la mémoire vive RAM : dans cet exemple, il s'agit de la sous-routine d'initialisation dont l'adresse de départ est 03C1 (page 24 du listing source). L'adresse de destination est 1900 H et le programme de transfert a son origine en 1800 H.

Intuitivement, vous voudriez charger la case 1900 H (destination) avec le contenu de la case 03C1. Puis recommencer l'opération après avoir incrémenté les deux adresses, ce qui permettrait d'effectuer la recopie du contenu de 03C2 dans 1901 et ainsi de suite jusqu'à ce que les 18 octets du bloc soient écrits dans la RAM.

Dans l'état actuel de nos connaissances, ce n'est pas possible même avec l'adressage par registres, puisque la source ou la destination est nécessairement le registre A.

Il nous faut d'abord charger A [chemin (1)] avec l'octet à transférer (figure 103) en utilisant l'instruction LD A,(BC) par exemple (ce qui implique que BC contienne 03C1) puis de transférer le contenu de A [(2)], avec l'instruction LD (DE), A (ce qui implique que DE contienne 1900).

Ceci étant fait, on incrémente BC qui pointe sur l'octet suivant (03C2) ainsi

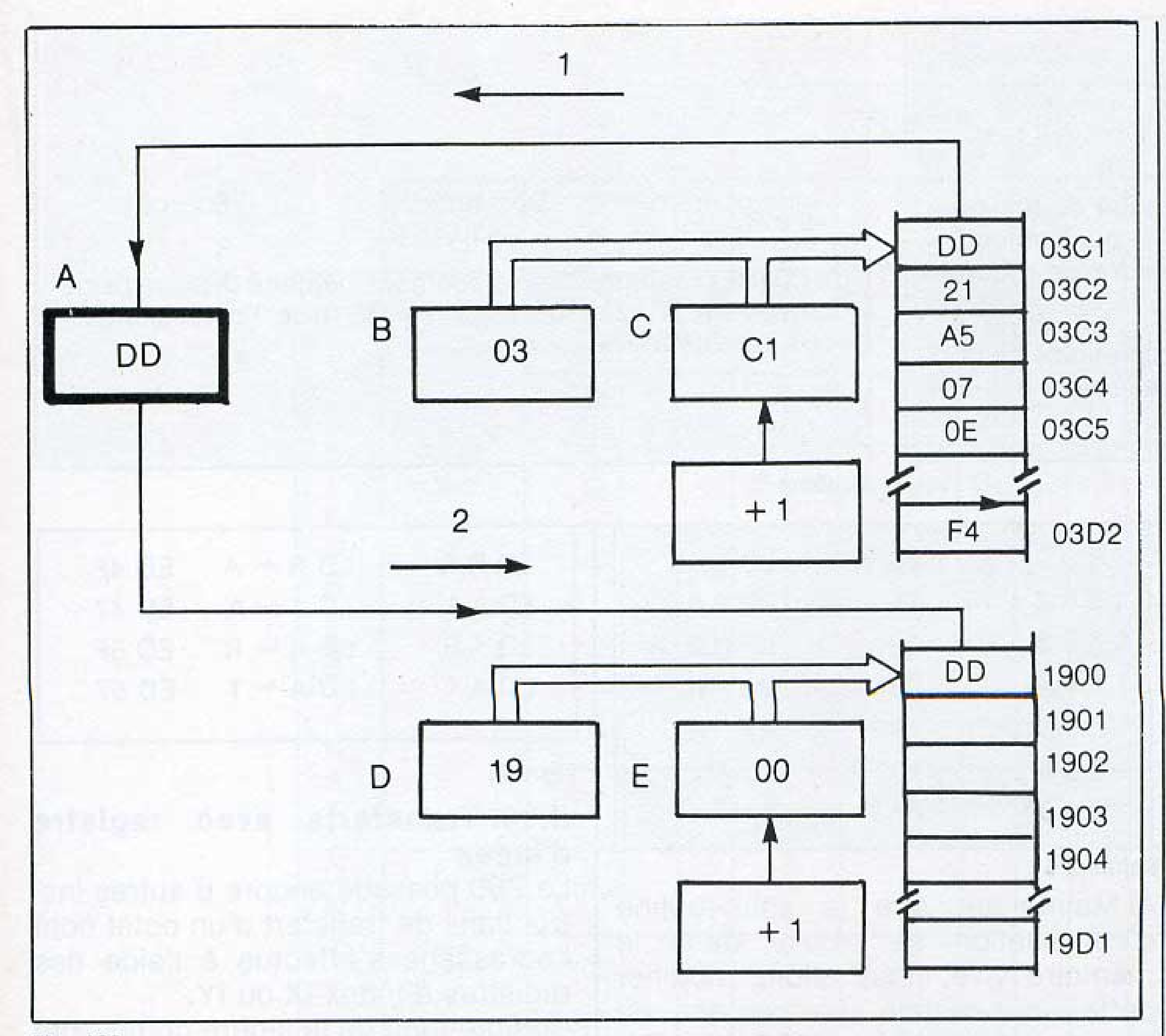


Fig. 103

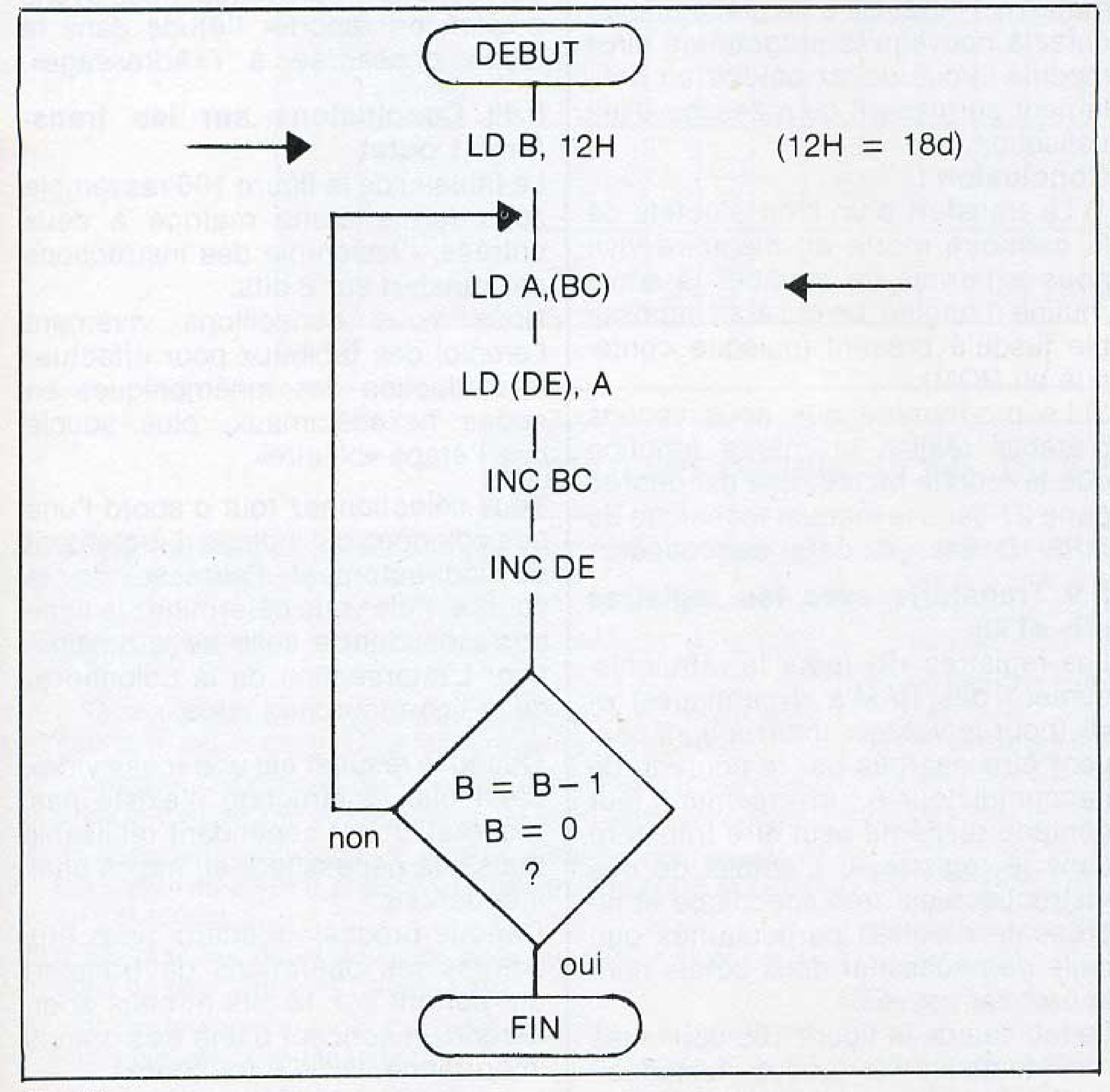


Fig. 104

que DE qui adresse l'emplacement 1901. En répétant 18 fois l'opération, le bloc sera recopié.

C'est ce que traduit l'organigramme de la figure 104 dans lequel apparaît l'instruction DJNZ que nous avons décrit dans LED-MICRO n° 10 page 62.

Malheureusement, le registre B est utilisé à la fois comme pointeur d'adresses dans l'instruction LD A,(BC) et comme décompteur de boucles (instruction DJNZ).

Sommes-nous dans une impasse ? Non, tant s'en faut. Une première solution (il y en a d'autres) est d'utiliser l'instruction de transfert avec adressage par la paire de registres HL.

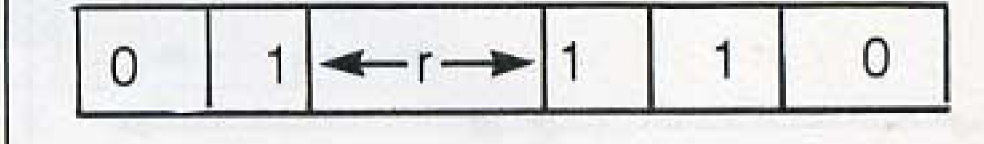
II.7. Transfert avec adressage par (HL)

Dans les quatre instructions de transfert avec adressage par le contenu de l'une des paires de registres BC ou DE, le registre A intervenait IMPLICITEMENT soit comme **élément source** soit comme **destination.**

Dans les deux types d'instructions de transfert avec adressage par (HL), le registre utilisé n'est plus exclusivement l'accumulateur **mais l'un des sept registres** (A ou B, C,... H ou L).

Les mnémoniques des instructions sont indiquées dans le schéma 6.

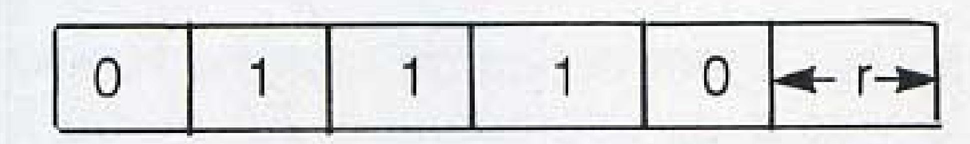
Le code binaire est :



avec : r :

1 1 1	A
0 0 0	B
0 0 1	C
0 1 0	D
0 1 1	E
1 0 0	H
1 0 1	L

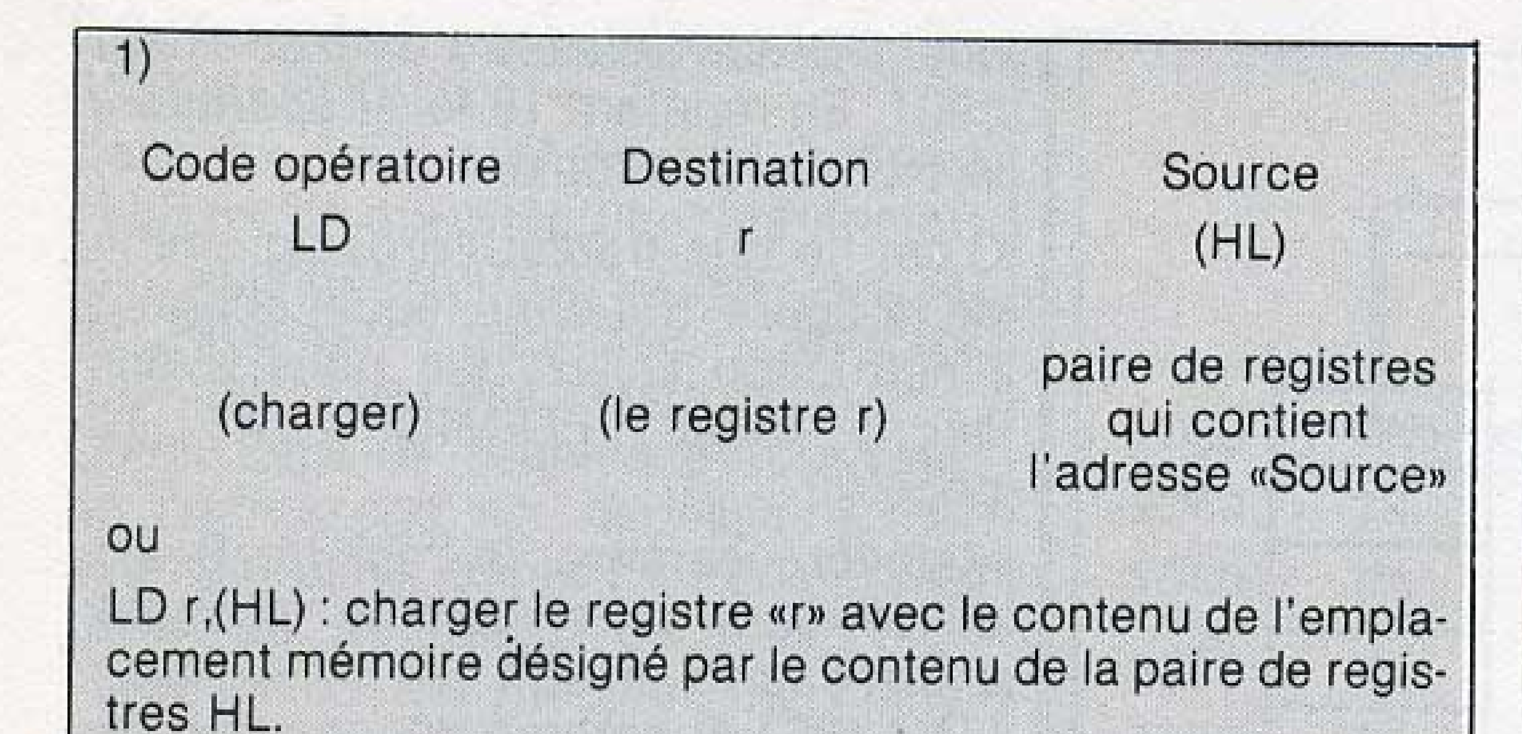
Le code binaire est :



même signification que r précédemment.

II.8. Exemples

1) Soit à charger le registre D avec


1)

Code opératoire	Destination	Source
LD	r	(HL)
(charger)	(le registre r)	paire de registres qui contient l'adresse «Source»

ou

LD r,(HL) : charger le registre «r» avec le contenu de l'emplacement mémoire désigné par le contenu de la paire de registres HL.

Schéma 6

2)

Code opératoire	Destination	Source
LD	(HL)	r

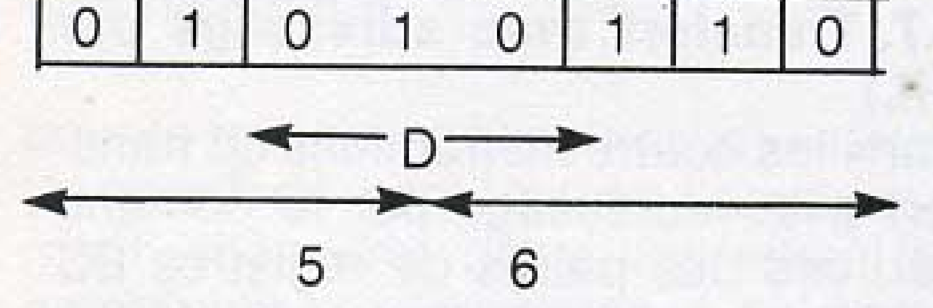
ou LD (HL),r : charger l'emplacement mémoire désigné par le contenu de la paire de registres HL avec l'octet contenu dans le registre «r».

Schéma 7

l'octet de l'emplacement adressé par la paire de registres HL.
Le code mnémonique est :

LD D,(HL)

et le code binaire est :

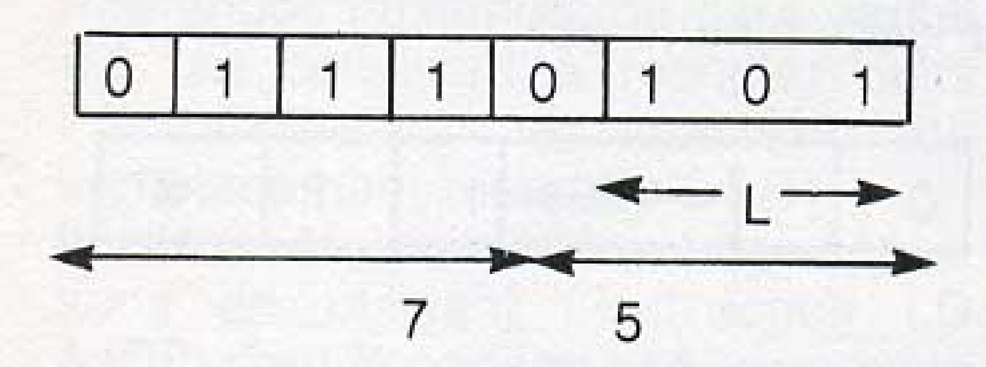


Ce qui donne pour le code hexadécimal : 56.

2) Soit à charger l'emplacement dont l'adresse est le contenu de la paire HL avec l'octet contenu dans le registre L, le code mnémonique est :

LD (HL), L

et le code binaire est :



ce qui donne pour le code hexadécimal : 75. (voir schéma 7) .
3) En remplaçant, dans l'exemple du transfert d'un bloc (fig. 104) Ld A,(BC) par Ld A,(HL) et en chargeant HL avec 03C1 d'une part et en incrémentant HL (code 23) au lieu de BC, nous obtenons le programme complet pour le transfert d'un bloc de données.
Etablissez le programme, introduisez-le et exécutez-le.

Nota : L'instruction de test est 10 et la valeur du déplacement FA ce qui donne 10 FA.

Si c'est correct, vous devez voir apparaître le message UPF-1 comme lors d'une mise sous tension. Voici la solution (voir schéma 8).

1800	06 12	LD B,12
1802	7E	LD A,(HL)
1803	12	LD (DE),A
1804	23	INC HL
1805	13	INC DE
1806	10 FA	DJNZ, −5
1808	F7	FIN

Schéma 8

4) Maintenant que la sous-routine d'initialisation se trouve dans la mémoire vive, nous allons modifier cette sous-routine. Le contenu du dernier octet (adresse : 19D1) est F4, mettez-y l'octet EE à sa place et exécutez à nouveau le programme ainsi modifié : vous devez obtenir un défilement permanent du message d'initialisation.

Conclusion :

1) Le transfert d'un bloc d'octets de la mémoire morte en mémoire vive nous a permis de modifier la sous-routine d'origine, ce qui était impossible jusqu'à présent (puisque contenue en ROM).
2) Le programme que nous venons d'établir réalise la même fonction que la touche MOVE. Elle est décrite page 27 dans le manuel technique du MPF-1B. Essayez cette commande.

II.9. Transferts avec les registres «R» et «I»

Les registres «R» (pour le rafraichissement des RAM's dynamiques) et «I» (pour le vecteur Interruption) peuvent être chargés par le contenu de l'accumulateur A ; inversement, leur contenu respectif peut être transféré dans le registre A. L'emploi de ces instructions est très spécifique et ne présente d'autres particularités que celle de nécessiter deux octets dont le premier est «ED».
Le tableau de la figure 105 donne les mnémoniques et codes hexadécimaux pour ces quatre instructions.

LD R,A	LD R ← A	ED 4F
LD 1,A	LD 1 ← A	ED 47
LD A,R	LD A ← R	ED 5F
LD A,1	LD A ← 1	ED 57

Fig. 105

II.10. Transferts avec registre d'Index

Le Z80 possède encore 6 autres instructions de transfert d'un octet dont l'adressage s'effectue à l'aide des registres d'Index IX ou IY.
Compte tenu de la légère complexité de ce mode d'adressage, nous avons préféré en reporter l'étude dans la leçon consacrée à l'«Adressage».

II.11. Conclusions sur les transferts 1 octet

Le tableau de la figure 106 rassemble sous forme d'une matrice à deux entrées, l'ensemble des instructions de transfert sur 8 bits.
Nous vous conseillons vivement l'emploi des tableaux pour effectuer la traduction des mnémoniques en codes hexadécimaux, plus souple que l'étape «binaire».

Vous sélectionnez tout d'abord l'une des colonnes qui indique directement ou indirectement l'adresse de la source. Puis vous déterminez la ligne correspondant à celle de la destination. L'intersection de la colonne et de la ligne fournit le code.

Quand le résultat est une «case vide», c'est que l'instruction n'existe pas. L'opération est cependant réalisable mais elle nécessitera au moins deux instructions.
Dans le prochain numéro, nous étudierons les opérations de transfert qui portent sur 16 bits et nous aborderons un concept d'une très grande importance, la PILE (ou Stack).

Philippe Duquesne

			SOURCE													
			Imp.		Registre							Reg. Ind.			Adr. éten	Imm
	Mode d'adress.		I	R	A	B	C	D	E	H	L	(BC)	(DE)	(HL)	(nn)	n
DESTINATION	Registre	A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	0A	1A	7E	3A n n	3E n
		B			47	40	41	42	43	44	45			46		06 n
		C			4F	48	49	4A	4B	4C	4D			4E		0E n
		D			57	50	51	52	53	54	55			56		16 n
		E			5F	58	59	5A	5B	5C	5D			5E		1E n
		H			67	60	61	62	63	64	65			66		26 n
		L			6F	68	69	6A	6B	6C	6D			6E		26 n
	Reg. indir.	(BC)			02											
		(DE)			12											
		(HL)			77	70	71	72	73	74	75					36 n
	Adresse étendue	(nn)			32 n n											
	Impl.	I			ED 47											
		R			ED 4F											

Fig. 106

REPERTOIRE INSTRUCTIONS DU Z-80R

Notations utilisées dans la description en mnémoniques du répertoire d'instructions du Z80 :

1) r désigne n'importe lequel des registres suivants : A, B, C, D, E, H, L.
2) (HL) désigne le contenu d'un emplacement mémoire dont l'adresse est le contenu de la paire de registres HL.
3) n désigne une donnée de un octet, dont la valeur est comprise entre 0 et 255 en décimal ou 00 et FF en hexadécimal.
4) nn désigne une quantité (donnée ou adresse) dont la valeur est comprise entre 0 et 65535 en décimal ou 0000 et FFFF en hexadécimal.
5) d désigne une quantité sur un octet en valeur algébrique, donc compris entre − 128 et + 127.
6) (nn) désigne le contenu d'un emplacement mémoire dont l'adresse est nn (2 octets).
7) b désigne une quantité de 0 à 7.
8) e désigne un déplacement sur un octet en valeur algébrique donc compris entre − 126 et + 129.
9) cc indique l'état des indicateurs pour les instructions conditionnelles JR, JP; CALL et RET.
10) qq désigne n'importe quelle paire de registres BC, DE, HL ou AF.
11) ss désigne l'une quelconque des paires de registres suivantes : BC, DE, HL et SP.
12) pp désigne l'une quelconque des paires de registres suivantes : BC, DE, IX et SP.
13) rr désigne l'une quelconque des paires de registres suivantes : BC, DE, IY et SP.
14) s désigne l'un des opérandes suivants : r, n, (HL), (IX + d), (IY + d).
15) dd désigne l'une des paires de registres suivants : BC, DE, HL et SP.
16) m désigne l'un des opérandes suivants : r, (HL), (IX + d), (IY + d).

Code mnémonique	Opération
ADC HL, ss	Additionner HL et le registre double ss avec le report. Résultat dans HL.
ADC A, s	Additionner l'accumulateur et l'opérande s avec le report. Résultat dans A.
ADD A, n	Additionner l'accumulateur et la valeur n. Résultat dans A.
ADD A, r	Additionner l'accumulateur et le registre r. Résultat dans A.
ADD A, (HL)	Additionner l'accumulateur et l'emplacement mémoire dont l'adresse est le contenu de HL. Résultat dans A.
ADD A, (IX + d)	Additionner l'accumulateur et l'emplacement mémoire dont l'adresse est le contenu de IX + d. Résultat dans A.
ADD A, (IY + d)	Additionner l'accumulateur et l'emplacement mémoire dont l'adresse est le contenu de IY + d. Résultat dans A.
ADD HL, ss	Additionner HL et le registre double ss. Résultat dans HL.
ADD IX, pp	Additionner IX et le registre double pp. Résultat dans IX.
ADD IY, rr	Additionner IY et le registre double rr. Résultat dans IY.
AND s	ET logique entre l'accumulateur et l'opérande s. Résultat dans A.
BIT b, (HL)	Test du bit b de l'emplacement mémoire dont l'adresse est le contenu de HL. Résultat dans Z (indicateur).
BIT b, (IX + d)	Test du bit b de l'emplacement mémoire dont l'adresse est le contenu de IX + d. Résultat dans Z (indicateur).
BIT b, (IY + d)	Test du bit b de l'emplacement mémoire dont l'adresse est le contenu de IY + d. Résultat dans Z (indicateur).
BIT b, r	Test du bit b du registre r. Résultat dans Z (indicateur).
CALL cc, nn	Saut à la sous-routine d'adresse nn si la condition cc est vraie.
CALL nn	Saut inconditionnel à la sous-routine d'adresse Nn.
CCF	Complémentation de l'indicateur C. Résultat dans C.
CP s	Comparaison de l'opérande s et de l'accumulateur. Résultat positionnement des indicateurs.
CPD	Comparaison du contenu d'emplacement mémoire pointé par HL avec l'accumulateur. HL et BC sont décrémentés. Résultat positionnement des indicateurs.
CPDR	Comparaison du contenu d'emplacement mémoire pointé par HL avec l'accumulateur. HL et BC sont décrémentés. l'opération est répétée jusqu'à ce que BC = 0. Résultat positionnement des indicateurs.
CPI	Comparaison du contenu d'emplacement mémoire pointé par HL avec l'accumulateur. HL est incrémenté et BC décrémenté. Résultat positionnement des indicateurs.

Code mnémonique	Opération
CPIR	Comparaison du contenu d'emplacement mémoire pointé par HL avec l'accumulateur. HL est incrémenté et BC décrémenté. L'opération est répétée jusqu'à ce que BC = 0. Résultat positionnement des indicateurs.
CPL	Complémentation à 1 de l'accumulateur. Résultat dans l'accumulateur.
DAA	Ajustement décimal de l'accumulateur.
DEC m	Décrémentation de l'opérande m.
DEC IX	Décrémentation du registre IX.
DEC IY	Décrémentation du registre IY.
DEC ss	Décrémentation du registre double ss.
DI	Interdiction des interruptions.
DJNZ e	Décrémentation du registre B et saut relatif de e si B est égal à 0.
EI	Autorisation des interruptions.
EX (SP), HL	Echange du registre HL avec le sommet de la pile.
EX (SP), IX	Echange du registre IX avec le sommet de la pile.
EX (SP), IY	Echange du registre IY avec le sommet de la pile.
EX AF, A'F'	Echange entre les registres AF et A'F'.
EX DE, HL	Echange entre les registres DE et HL.
EXX	Echange entre les registres BC, DE et HL avec le contenu respectivement des registres B'C', D'E' et H'L'.
HALT	Arrêt du CPU (attente d'une interruption ou RAZ).
IMO	Sélection du mode d'interruption 0.
IM1	Sélection du mode d'interruption 1.
IM2	Sélection du mode d'interruption 2.
IN A, (n)	Chargement de l'accumulateur à partir du contenu du port d'adresse n.
IN r, (C)	Chargement du registre r à partir du contenu du port dont l'adresse est le contenu de C.
INC (HL)	Incrémentation de l'emplacement mémoire pointé par HL.
INC IX	Incrémentation de IX.
INC (IX + d)	Incrémentation de l'emplacement mémoire pointé par IX + d.
INC IY	Incrémentation de IY.
INC (IY + d)	Incrémentation de l'emplacement mémoire pointé par IY + d.
INC r	Incrémentation du registre r.
INC ss	Incrémentation de la paire de registres ss.
IND	Chargement de l'emplacement pointé par HL avec le contenu du port dont l'adresse et le contenu de C, décrémentation de HL et de B.
INDR	Chargement de l'emplacement pointé par HL avec le contenu du port dont l'adresse est le contenu de C, décré-

Code mnémonique	Opération
	mentation de HL et de B. L'opération est répétée jusqu'à ce que B = 0.
INI	Chargement de l'emplacement pointé par HL avec le contenu du port dont l'adresse est le contenu de C, incrémentation de HL et décrémentation de B.
INIR	Chargement de l'emplacement pointé par HL avec le contenu du port dont l'adresse est le contenu de C, incrémentation de HL et décrémentation de B. L'opération est répétée jusqu'à ce que B = 0.
JP (HL)	Saut inconditionnel à l'adresse contenue dans HL.
JP (IX)	Saut inconditionnel à l'adresse contenue dans IX.
JP (IY)	Saut inconditionnel à l'adresse contenue dans IY.
JP cc, nn	Saut à l'adresse nn, si la condition cc est vraie.
JP nn	Saut inconditionnel à l'adresse nn.
JR C, e	Saut à l'adresse PC + e si l'indicateur C = 1.
JR e	Saut inconditionnel à l'adresse PC + e.
JR NC, e	Saut à l'adresse PC + e si l'indicateur C = 0.
JR NZ, e	Saut à l'adresse PC + e si l'indicateur Z = 0
JR Z, e	Saut à l'adresse PC + e si l'indicateur Z = 1.
LD A, (BC)	Chargement de l'accumulateur avec le contenu de l'emplacement mémoire dont l'adresse est le contenu de BC.
LD A, (DE)	Chargement de l'accumulateur avec le contenu de l'emplacement mémoire dont l'adresse est le contenu de DE.
LD A, I	Chargement de l'accumulateur avec le contenu du registre I.
LD A, (nn)	Chargement de l'accumulateur avec le contenu de l'adresse nn.
LD A, R	Chargement de l'accumulateur avec le contenu du registre R.
LD (BC), A	Chargement de l'emplacement mémoire dont l'adresse est le contenu de BC avec celui de A.
LD (DE), A	Chargement de l'emplacement mémoire dont l'adresse est le contenu de DE avec celui de A.
LD (HL), n	Chargement de l'emplacement mémoire dont l'adresse est le contenu de HL avec la donnée n.
LD dd, nn	Chargement de la paire de registres dd avec la donnée nn.
LD dd, (nn)	Chargement de la paire de registres dd avec la donnée contenue aux adresses nn et nn + 1.
LD HL, (nn)	Chargement de HL avec la donnée contenue aux adresses nn et nn + 1.
LD (HL), r	Chargement de l'emplacement mémoire

Code mnémonique	Opération
	dont l'adresse est le contenu de HL avec celui du registre r.
LD I, A	Chargement du registre I avec le contenu de A.
LD IX, nn	Chargement du registre IX avec nn.
LD IX, (nn)	Chargement du registre IX avec la donnée contenue aux adresses nn et nn + 1.
LD (IX + d), n	Chargement de l'emplacement mémoire d'adresse IX + d avec la donnée n.
LD (IX + d), r	Chargement de l'emplacement mémoire d'adresse IX + d avec le contenu du registre r.
LD IY, nn	Chargement du registre IY avec nn.
LD IY, (nn)	Chargement du registre IY avec la donnée contenue aux adresses nn et nn + 1.
LD (IY + d), n	Chargement de l'emplacement mémoire d'adresse IY + d avec la donnée n.
LD (IY + d), r	Chargement de l'emplacement mémoire d'adresse IY + d avec le contenu du registre r.
LD (nn), A	Chargement de l'emplacement mémoire d'adresse nn avec le contenu de A.
LD (nn), dd	Chargement des emplacements mémoires nn et nn + 1 avec le contenu de la paire de registres dd.
LD (nn), HL	Chargement des emplacements mémoires nn et nn + 1 avec le contenu de HL.
LD (nn), IX	Chargement des emplacements mémoires nn et nn + 1 avec le contenu de IX.
LD (nn), IY	Chargement des emplacements mémoires nn et nn + 1 avec le contenu de IY.
LD R, A	Chargement du registre R avec le contenu de A.
LD r, (HL)	Chargement du registre r avec le contenu de l'emplacement mémoire dont l'adresse est le contenu de HL.
LD r, (IX + d)	Chargement du registre r avec le contenu de l'emplacement mémoire dont l'adresse est le contenu de IX + d.
LD r, (IY + d)	Chargement du registre r, avec le contenu de l'emplacement mémoire dont l'adresse est le contenu de IY + d.
LD r, n	Chargement du registre r avec la donnée n.
LD r, r'	Chargement du registre r avec le contenu de r'.
LD SP, HL	Chargement du registre SP avec le contenu de HL.
LD SP, IX	Chargement du registre SP avec le contenu de IX.
LD SP, IY	Chargement du registre SP avec le contenu de IY.
LDD	Chargement de l'emplacement d'adresse DE avec le contenu de

Code mnémonique	Opération
	l'emplacement d'adresse HL. Décrémentation de DE, HL et BC.
LDDR	Chargement de l'emplacement d'adresse DE avec le contenu de l'emplacement d'adresse HL. Décrémentation de DE, HL et BC. L'opération est répétée jusqu'à ce que BC = 0.
LDI	Chargement de l'emplacement d'adresse DE avec le contenu de l'emplacement d'adresse HL. Incrémentation de DE et HL. Décrémentation de BC.
LDIR	Chargement de l'emplacement d'adresse DE avec le contenu de l'emplacement d'adresse HL. Incrémentation de DE et HL. Décrémentation de BC. L'opération est répétée jusqu'à ce que BC = 0.
POP IX	Chargement de IX avec le sommet de la pile.
POP IY	Chargement de IY avec le sommet de la pile.
POP qq	Chargement de la paire de registres qq avec la sommet de la pile.
PUSH IX	Chargement du sommet de la pile avec IX.
PUSH IY	Chargement du sommet de la pile avec IY.
PUSH qq	Chargement du sommet de la pile avec le contenu de la paire de registres qq.
RES b, m	Remise à zéro du bit b de l'opérande m.
RET	Retour d'un sous-programme.
RET cc	Retour d'un sous-programme si la condition cc est vraie.
RETI	Retour d'un sous-programme d'interruption.
RETN	Retour d'un sous-programme d'interruption non masquable.
RL m	Rotation à gauche de l'opérande m au travers de l'indicateur C.
RLA	Rotation à gauche de l'accumulateur au travers de l'indicateur C.
RLC (HL)	Rotation à gauche du contenu de l'emplacement mémoire d'adresse HL. Recopie du bit de poids fort dans l'indicateur C.
RLC (IX + d)	Rotation à gauche du contenu de l'emplacement mémoire d'adresse (IX + d). Recopie du bit de poids fort dans l'indicateur C.
RLC (IY + d)	Rotation à gauche du contenu de l'emplacement mémoire d'adresse (IY + d). Recopie du bit de poids fort dans l'indicateur C.
NEG	Complément à 2 du contenu de l'accumulateur. Résultat dans A.
NOP	Pas d'opération.
OR s	Opération «OU logique» entre l'opérande s et le contenu de A. Résultat dans A.

Code mnémonique	Opération
OTDR	Chargement du port dont l'adresse est le contenu de C avec le contenu de l'emplacement d'adresse HL. Décrémentation de HL et de BC. L'opération est répétée jusqu'à ce que BC = 0.
OTIR	Chargement du port dont l'adresse est le contenu de C avec le contenu de l'emplacement d'adresse HL. Incrémentation de HL et décrémentation de BC. L'opération est répétée jusqu'à ce que BC = 0.
OUT (C), r	Chargement du port dont l'adresse est le contenu de C avec le contenu du registre r.
OUT (n), A	Chargement du port d'adresse n avec le contenu de A.
OUT D	Chargement du port dont l'adresse est le contenu de C avec le contenu de l'emplacement d'adresse HL. Décrémentation de HL et BC.
OUT I	Chargement du port dont l'adresse est le contenu de C avec le contenu de l'emplacement d'adresse HL. Incrémentation de HL et décrémentation de BC.
RLC r	Rotation à gauche du contenu du registre r. Recopie du bit de poids fort dans l'indicateur C.
RLCA	Rotation à gauche du contenu de l'accumulateur. Recopie du bit de poids fort dans l'indicateur C.
RLD	Rotation décimale à gauche.
RR m	Rotation à droite de l'opérande m au travers de l'indicateur C.
RRA	Rotation à droite du contenu de l'accumulateur au travers de l'indicateur C.
RRC m	Rotation à droite de l'opérande m. Recopie du bit de poids faible dans l'indicateur C.
RRCA	Rotation à droite du contenu de l'accumulateur. Recopie du bit de poids faible dans l'indicateur C.
RRD	Rotation décimale à droite.
RST p	Saut à l'adresse p.
SBC A, s	Soustraire de l'accumulateur l'opérande s ainsi que le report. Résultat dans A.
SBC HL, ss	Soustraire de HL le registre double ss avec le report. Résultat dans HL.
SCF	Mise à 1 de l'indicateur C.
SET b, (HL)	Mise à 1 du bit b de l'emplacement mémoire dont l'adresse est le contenu du HL.
SET b, (IX + d)	Mise à 1 du bit b de l'emplacement mémoire dont l'adresse est le contenu de IX + d.
SET b, (IY + d)	Mise à 1 du bit b de l'emplacement mémoire dont l'adresse est le contenu de IY + d.

Code mnémonique	Opération	Code mnémonique	Opération
SET b, r	Mise à 1 du bit b du registre r.		m. Le bit de poids faible est dans l'indicateur C.
SLA m	Décalage arithmétique à gauche de l'opérande m. Le bit de poids fort est dans l'indicateur C.	SUB s	Soustraire l'opérande s de l'accumulateur. Résultat dans l'accumulateur.
SRA m	Décalage arithmétique à droite de l'opérande m. Le bit de poids faible est dans l'indicateur C.	XOR s	OU exclusif logique entre l'accumulateur et l'opérande s. Résultat dans l'accumulateur.
SRL m	Décalage logique à droite de l'opérande		

EXERCICES

Nota :

1. Dans les exercices qui suivent, vous pouvez être amenés à utiliser l'instruction DJNZ et donc de devoir effectuer le calcul d'une adresse relative. Votre travail peut être grandement facilité en utilisant la commande RELA décrite dans le Manuel Technique du MPF-I, pages 31 et 32.
2. Terminez vos programmes avec une instruction FIN (F7).

Exercice 1

Etablir le code binaire et en déduire le code hexadécimal pour les opérations de transferts suivantes. Vérifiez avec le tableau de la figure 106.

a) LD A,C
b) LD A,H
c) LD D,A
d) LD B,E
e) LD L,B

Exercice 2

Expliciter les codes hexadécimaux suivants :

a) 3A 0F 18
b) 32 0F 18
c) 3É FE
d) ED 5F
e) 2E 4F

Exercice 3

Quelle différence existe-t-il (s'il en existe une) entre les deux instructions suivantes :

LD A, BC
LD A, (BC)

Indiquez les codes hexadécimaux correspondants.

Exercice 4

Même question entre les instructions suivantes :

LD DE, A
LD (DE), A

Exercice 5

Déterminez le code hexadécimal des instructions suivantes :

a) LD C, (HL)
b) LD A, BE
c) LD A, (BE)
d) LD A, 1900
e) LD I, (HL)
f) LD A, (1900)
g) LD HL, L
h) LD C, (DE)
j) LD BC, A
k) LD R, A

Remarques.

Exercice 6

Ecrire un programme à l'aide des instructions de transfert sur 8 bits qui permet de charger les paires de registres HL et DE avec les quantités respectives 03C1 et 1900. Reprendre l'ensemble de l'exercice II.8.3.

Exercice 7

A l'aide des instructions de transfert sur 8 bits, établissez le programme qui permet d'inverser le contenu de deux cases mémoires consécutives. Exemple :

AVANT Exécution

1950 [11]
1951 [22]

APRES Exécution

1950 [22]
1951 [11]

Exercice 8

Même question dans le cas de six cases consécutives.

AVANT Exécution

1950 [05]
1951 [04]
1952 [03]
1953 [02]
1954 [01]
1955 [00]

APRES Exécution

1950 [00]
1951 [01]
1952 [02]
1953 [03]
1954 [04]
1955 [05]

Exercice 9

a) Réaliser l'opération

LD L, (HL + 1)

b) Ecrire le programme qui réalise la séquence suivante :

LD L, (HL)
LD H, (HL + 1)

par exemple HL contient 1A00 et les cases mémoires 1A00 et 1A01 contiennent respectivement FF et 20.

COURS PRATIQUE DE MICROPROCESSEUR AVEC LE MICROPROFESSOR MPF-IB

SIXIEME PARTIE

Le langage du Z80R (2) (suite)

SOMMAIRE

III. TRANSFERT DE DONNEES (2 OCTETS)

III.1. Introduction

Le numéro précédent (LM n° 13) était consacré aux transferts de données qui ne comportaient qu'un seul octet. Nous allons étudier les transferts de données de deux octets (16 bits). Cette quantité est souvent désignée par le terme «mot».

Dans cette nouvelle série d'instructions de transfert, seuls les registres de taille 16 bits peuvent constituer l'élément source ou destinataire. Ce qui signifie que les paires de registres BC, DE et HL ou B'C', D'E' et H'L' sont considérées comme constituant un registre unique de 16 bits.

15	8 7	0
	A	F
	B	C
Registres 16 bits	D	E
	H	L
	Stack pointer	
	Index IX	
	Index IY	

Fig. 107

La figure 107 rappelle l'architecture interne des registres internes du CPU.

La paire de registres BC est constituée des registres B et C ; l'octet de poids fort étant contenu dans B, l'octet de poids faible dans C.

La paire de registres DE est constituée des registres D et E et la paire HL des registres H et L.

Le registre Stack Pointer ou SP et les deux registres index IX et IY sont à l'origine exclusivement de taille 16 bits (comme le PC qui ne figure pas).

Les deux registres A et F de 8 bits figurent en pointillés. Ils ne forment en aucun cas un registre 16 bits à part entière, cependant ils sont fréquemment associés l'un à l'autre. C'est dans cette configuration que nous étudierons une instruction de sauvegarde et une de restitution qui concernent le couple AF.

Nous n'avons pas fait figurer le registre PC dans la mesure où il n'est pas possible de charger ce registre par des instructions de type «LOAD».

Cependant il est possible d'en modifier le contenu, mais les instructions sont d'un type totalement différent (saut, appel, retour, etc.), que nous étudierons ultérieurement.

En résumé, le CPU dispose de six registres 16 bits pour effectuer des opérations sur des mots de deux octets.

Les opérations que nous rencontrerons sont du même type que celles sur 1 octet, à savoir :

— opérations de chargement immédiat

— opérations entre registres et mémoire

— opération de chargement d'un registre par un autre.

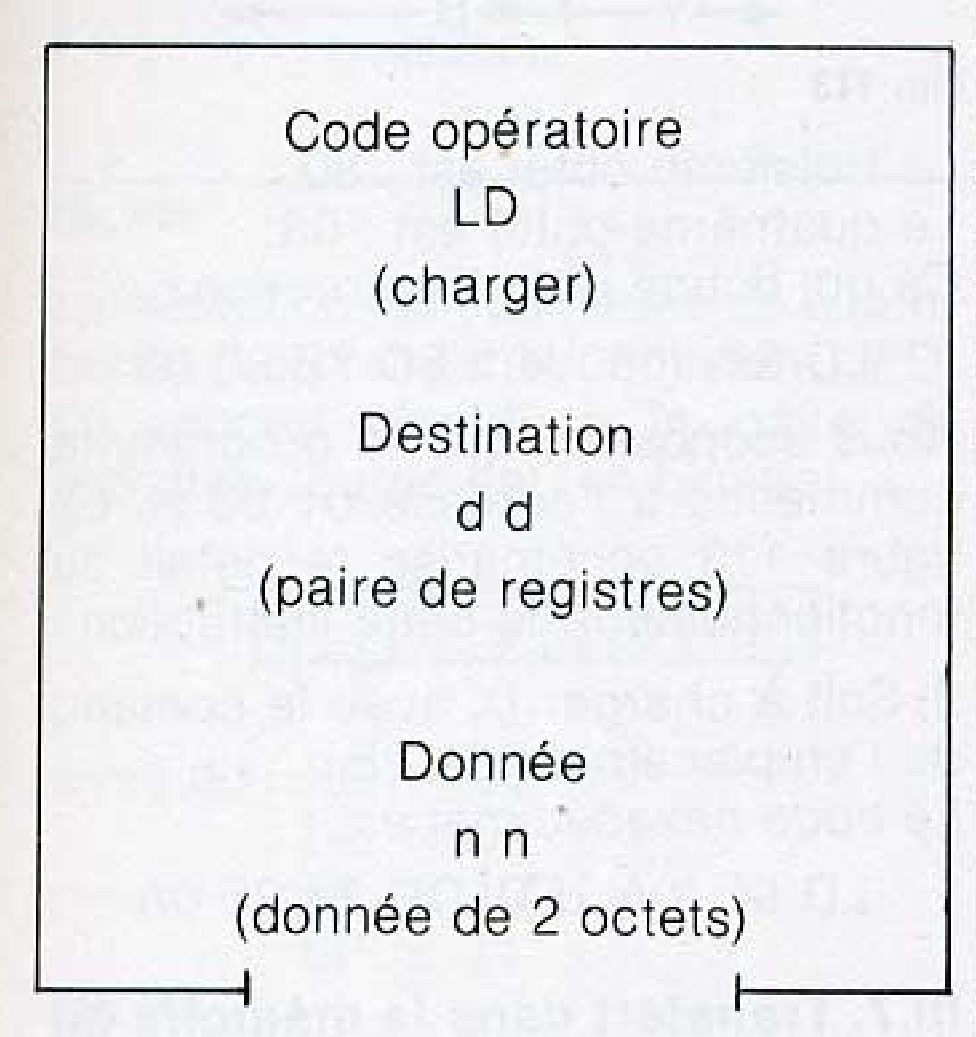
Nous introduirons un concept nouveau et fondamental qui confère au microprocesseur une très grande

souplesse d'utilisation : la PILE (ou STACK) et son registre associé, le pointeur de pile (STACK POINTER) ou registre SP.

III.2. Chargement immédiat

Le chargement immédiat d'une paire de registres est une opération identique au chargement immédiat d'un registre (LD r, n). La différence réside dans le fait que le **code opératoire n'est plus suivi de 1 mais de 2 octets** (fig. 108).

Le schéma de l'instruction est :



ou LD dd, nn : charger la paire de registres dd avec nn. L'octet qui suit immédiatement l'opérande (ou deuxième octet de l'instruction) constitue l'octet de poids faible.

Le code binaire est :

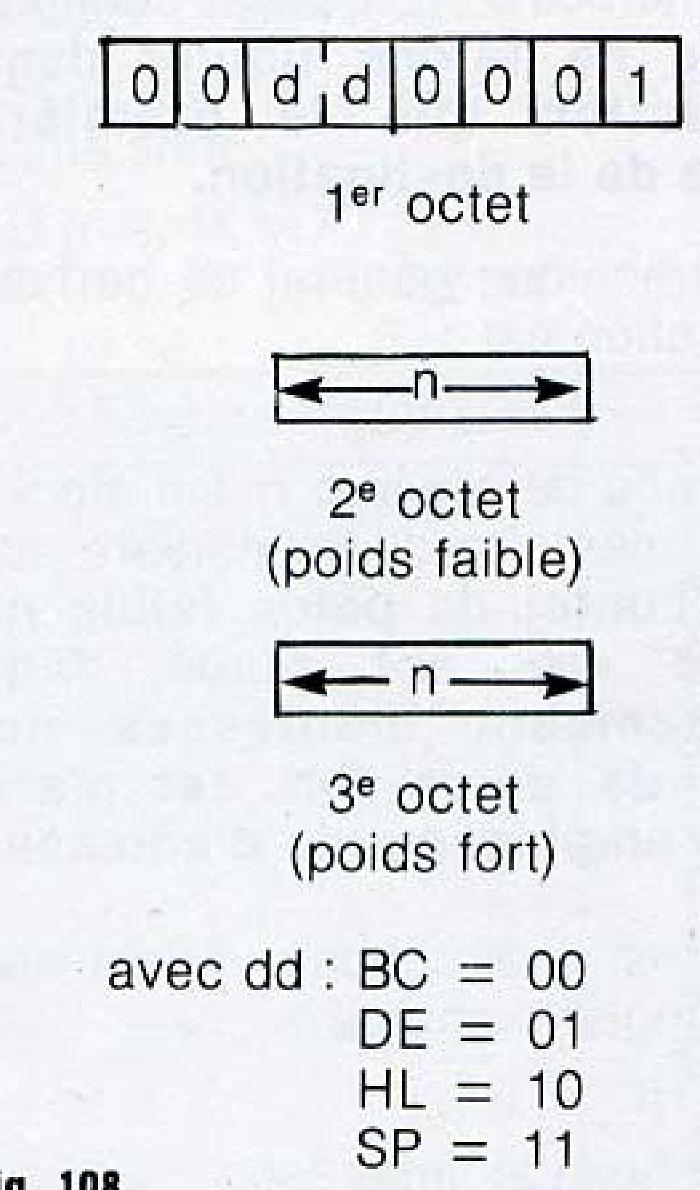


Fig. 108

III.3. Exemple

Soit à charger la paire de registres HL avec la quantité hexadécimale 1F 3E.

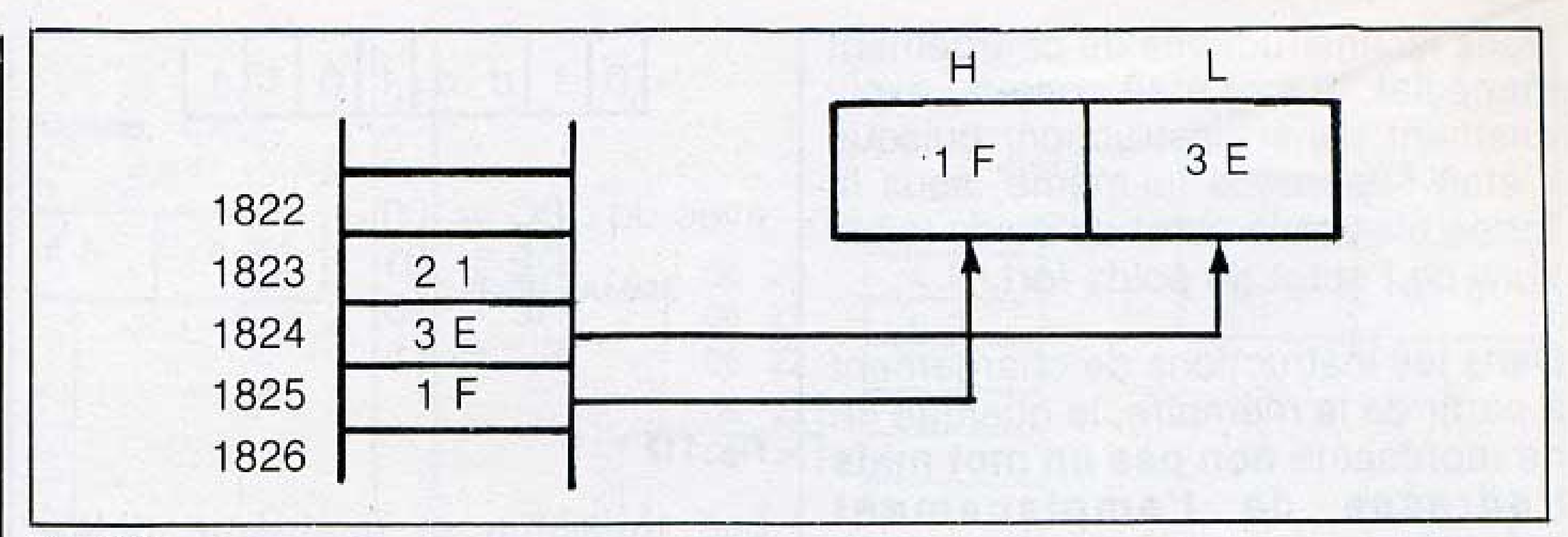


Fig. 110

Le code binaire du premier octet est (figure 109) :

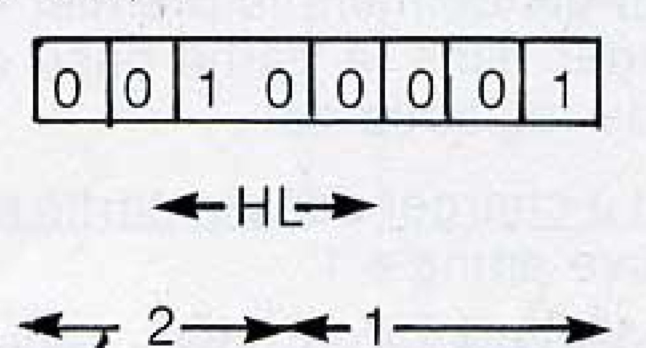


Fig. 109

Le code du deuxième octet est l'octet de poids faible : 3 E

Le code du troisième octet est l'octet de poids fort : 1 F

d'où LD HL, 1F 3E est 21 3E 1F.

La figure 110 schématise l'exécution de cette instruction supposée commencer en 1823 H.

Exercices :

Etablir les instructions qui permettent de réaliser les opérations suivantes :

a) Charger SP avec 08 10 H
b) Charger DE avec 00 F2 H
c) Charger BC avec 10 01 H

III.4. Chargement des registres IX et IY

Dans l'établissement du code binaire, comme le montre la figure 108, on ne dispose dans le premier octet de l'instruction que de deux bits (bits 5 et 6) qui permettent d'identifier quatre registres 16 bits parmi les six disponibles.

Les instructions qui réalisent le chargement des registres index IX et IY comportent un octet supplémentaire, qui est en quelque sorte un préfixe. La valeur est DD quand il s'agit du registre IX et FD pour le registre IY. Du fait de la présence du préfixe, la longueur des instructions n'est plus de trois mais de quatre octets.

Les codes mnémoniques et hexadécimaux sont :

LD IX, nn ou DD 21 n n
LD IY, nn ou FD 21 n n

Exemple :

Soit à charger le registre IY avec 0F 00H.

Le code hexadécimal correspondant est :

LD IY, 0F 00 FD 21 00 0F

Ceci peut se vérifier très aisément à l'aide de votre MPF-1B. Introduisez les quatre codes FD, 21... plus F7. Exécutez le programme et relisez le contenu de IY.

III.5. Chargement d'un registre 16 Bits à partir de la mémoire

La mémoire est organisée en octets, aussi pour obtenir un mot de 16 Bits, il faudra non pas utiliser un seul emplacement mais deux. Dans tous les cas, la case mémoire indiquée est celle qui contient l'octet de poids faible **étant implicite que l'octet de poids fort occupe l'emplacement suivant immédiatement** (figure 111).

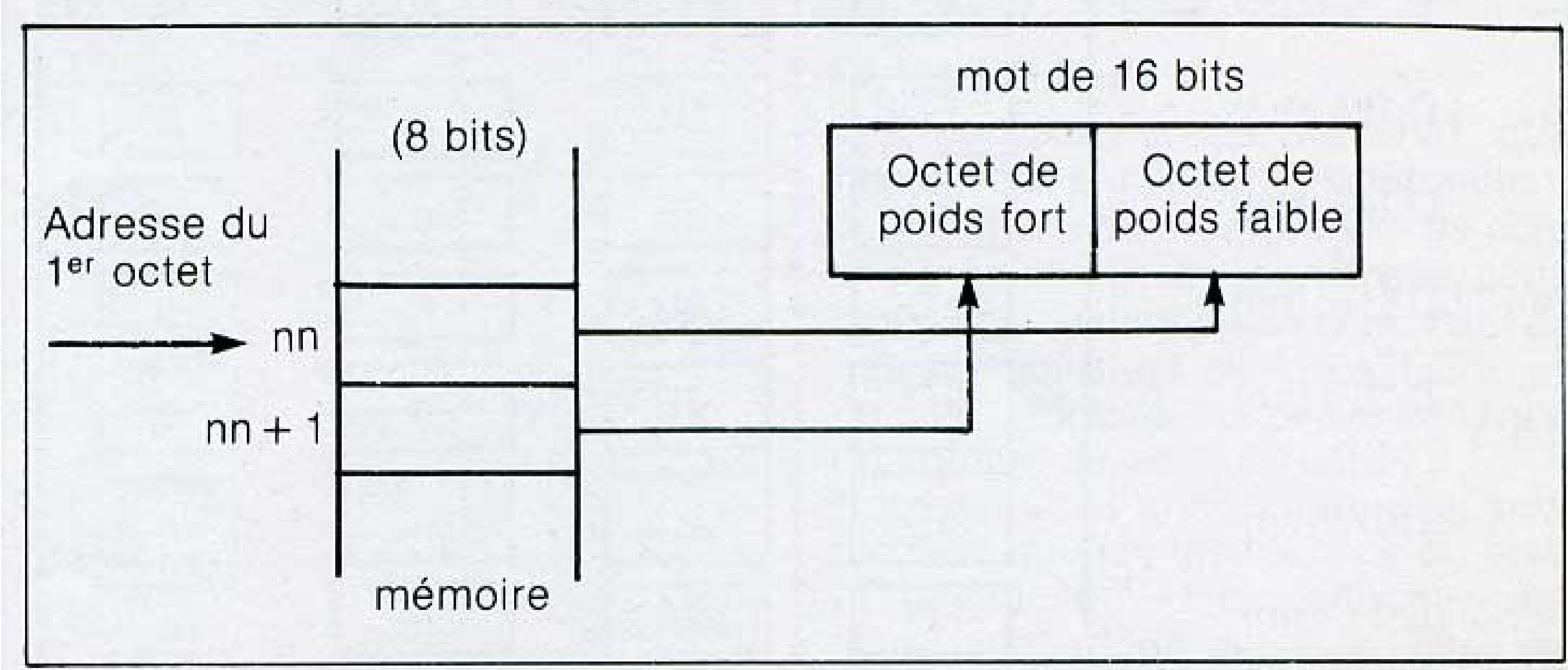


Fig. 111

Dans les instructions de chargement immédiat, le mot était contenu explicitement dans l'instruction, puisque c'était l'opérande lui-même, sous la forme classique octet de poids faible suivi de l'octet de poids fort.

Dans les instructions de chargement à partir de la mémoire, la quantité nn ne représente **non pas un mot mais l'adresse de l'emplacement mémoire** qui contient l'octet de poids faible, comme rappelé ci-dessus. Comme nn représente une case mémoire, l'emploi des parenthèses est indispensable.

Le premier transfert que nous examinerons est le chargement d'un registre 16 bits à partir de deux emplacements mémoire.
Le mnémonique général de cette instruction est :

LD rr, (nn)

Le contenu de l'emplacement mémoire (nn) constitue l'octet de poids faible et le contenu de l'emplacement suivant, nn + 1 constitue l'octet de poids fort.

a) rr = HL
le code hexadécimal de l'instruction est :

LD HL, (nn) : 2A n n

b) rr désigne l'une des paires de registres BC, DE, HL ou SP.
L'instruction est constituée de quatre octets. Le premier est constant : ED. Le second identifiant la paire de registres destinataire est (fig. 112) en binaire :

0	1	d	d	1	0	1	1

avec dd : BC = 00
DE = 01
HL = 10
SP = 11

Fig. 112

Les troisième et quatrième octets sont respectivement le poids faible et le poids fort de l'adresse de l'emplacement qui contient le premier octet à charger dans la partie base de la paire de registres dd.

L'octet à charger dans la partie haute se trouve en nn + 1.

c) rr désigne l'un des registres d'index IX ou IY

Les codes hexadécimaux correspondants sont :

LD IX, (nn) : DD 2A n n
LD IY, (nn) : FD 2A n n

Il s'agit aussi dans ce cas de deux instructions de 4 octets.

Le tableau I résume l'ensemble des instructions de chargement d'un registre 16 bits à partir de la mémoire.

Remarque : L'instruction LD HL, (nn) a deux équivalences hexadécimales (2A nn et ED 4B nn). Cependant, si le résultat est identique, la première est plus performante. Tout d'abord elle n'occupe que trois octets mémoire au lieu de quatre et sa durée d'exécution est plus rapide puisqu'elle ne nécessite que 16 temps d'horloge au lieu de 20.

Mnémonique	Codes hexadécimaux			
LD HL, (nn)	2A	n	n	
LD BC, (nn)	ED	4B	n	n
LD DE, (nn)	ED	5B	n	n
LD HL, (nn)	ED	6B	n	n
LD SP, (nn)	ED	7B	n	n
LD IX, (nn)	DD	2A	n	n
LD IY, (nn)	FD	2A	n	n

Tableau I

III.6. Exemples

1) Soit à charger le registre SP avec le contenu de l'emplacement 08 80 H. Supposons que la mémoire d'adresse 08 80 H contienne «00» et que l'emplacement suivant 08 81 H contienne «0F».
Le premier octet est : ED
Le deuxième octet est :

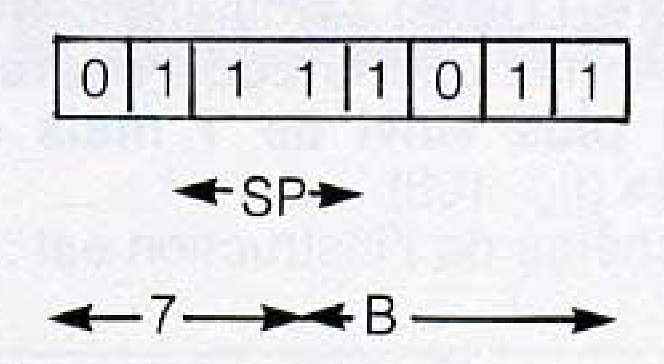


Fig. 113

Le troisième octet est : 80
Le quatrième octet est : 08.
Ce qui donne pour l'instruction :

LD SP, (880 H) : ED 7B 80 08

Nous supposons que le programme commence à l'adresse 01 00 H. La figure 113 schématise le détail du fonctionnement de cette instruction.

2) Soit à charger IX avec le contenu de l'emplacement 0A 2E.
Le code hexadécimal est :

LD IX, (0A 2E) : DD 2A 2E 0A

III.7. Transfert dans la mémoire du contenu d'un registre 16 bits

Le second transfert à étudier est le stockage dans la mémoire du contenu de l'un des registres 16 bits. Comme dans le cas précédent, ce contenu occupera deux emplacements mémoires (2 octets) contigus et **nous ne ferons figurer dans l'instruction que la première adresse de la destination.**

Le mnémonique général de ce type d'instruction est :

LD (nn), rr

Le contenu du registre rr est stocké dans la mémoire de la manière suivante : **l'octet de poids faible du registre «rr» est placé dans l'emplacement d'adresses nn. L'octet de poids fort est placé dans l'emplacement d'adresses nn + 1.**
Les codes «instruction» varient suivant le registre source :

a) rr = HL

Le code hexadécimal est :

LD (nn), HL : HL : 22 nn

b) rr désigne l'une des paires de registres BC, DE, HL ou SP.

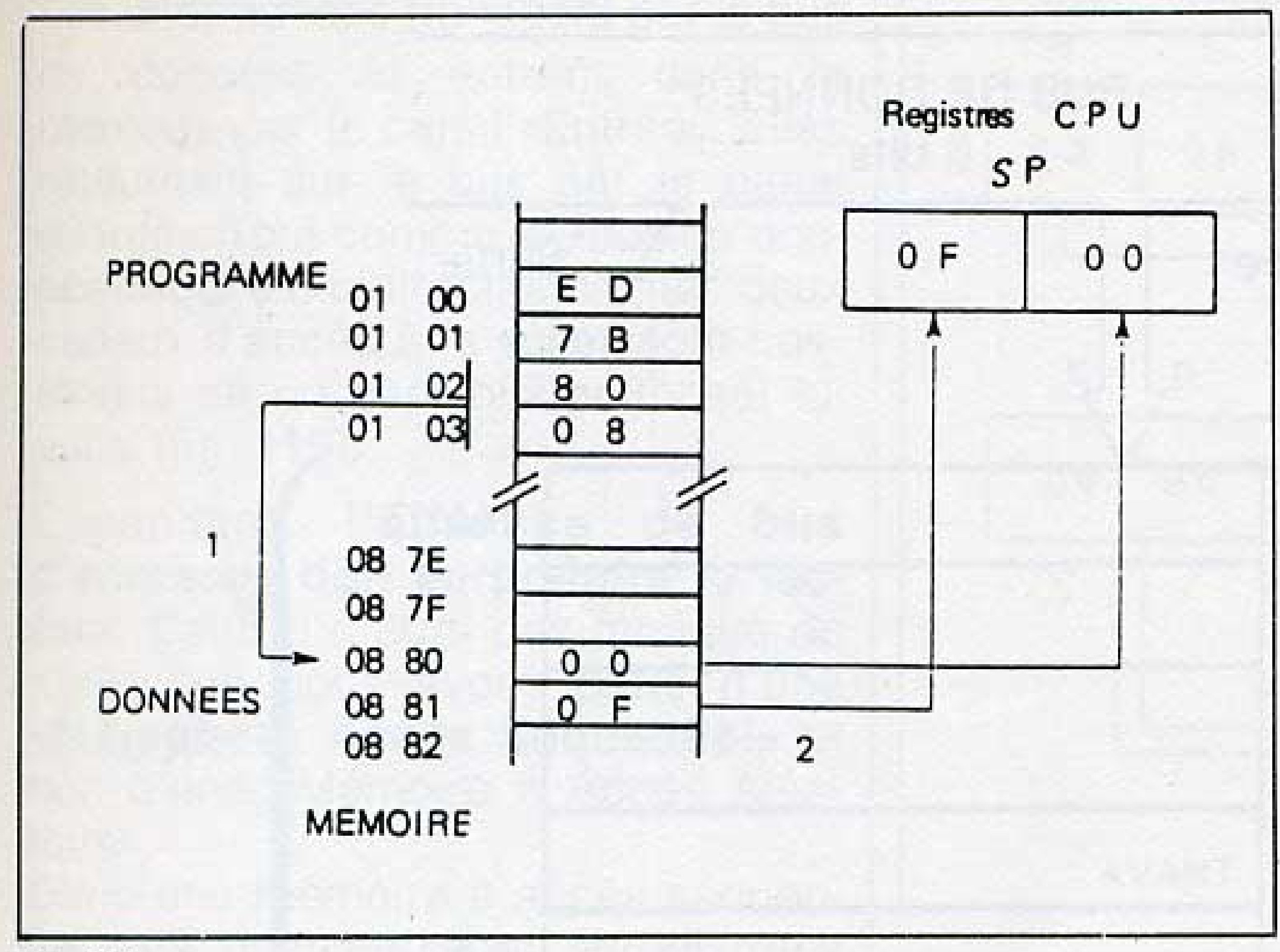


Fig. 114

L'instruction est constituée de quatre octets. Le premier est constant : ED. Le second identifiant la paire de registres source est (en binaire) :

0	1	d	d	0	0	1	1

avec dd : BC = 00
DE = 01
HL = 10
SP = 11

Fig. 115

Les troisième et quatrième sont respectivement le poids faible et le poids fort de l'adresse de l'emplacement destinataire, qui stockera la partie faible de la paire de registres rr.

c) rr désigne l'un des registres d'index IX ou IY.

Les codes hexadécimaux correspondants sont :

LD (nn), IX : DD 22 n n
LD (nn), IY : FD 22 nn

Le tableau II résume l'ensemble des instructions de chargement de la mémoire à partir d'un registre 16 bits.

III.8. Exemples

Soit à stocker le contenu du registre HL dans l'emplacement 3D 10H et suivant (3D 11H).

Nous supposons que HL contient 50 00 H.

Le premier octet est : ED.

Le second octet est :

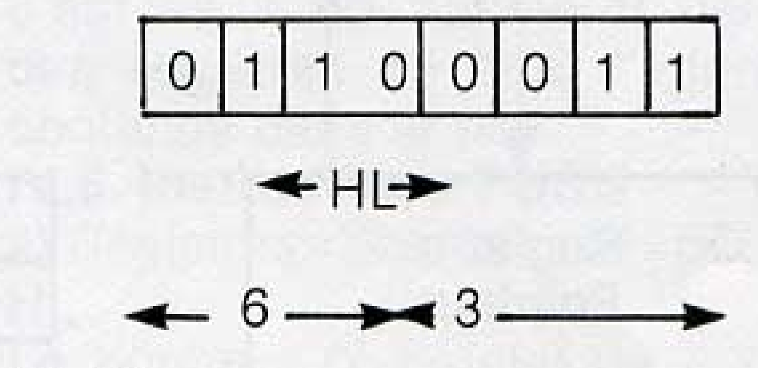


Fig. 116

Le troisième octet est : 10
Le quatrième octet est : 3D.
Ce qui donne pour l'instruction :
LD (3D 10), HL : ED 63 10 3D

Mnémonique	Codes hexadécimaux			
LD (nn), HL	22	n	n	
LD (nn), BC	ED	43	n	n
LD (nn), DE	ED	53	n	n
LD (nn), HL	ED	63	n	n
LD (nn), SP	ED	73	n	n
LD (nn), IX	DD	22	n	n
LD (nn), IY	FD	22	n	n

Tableau II

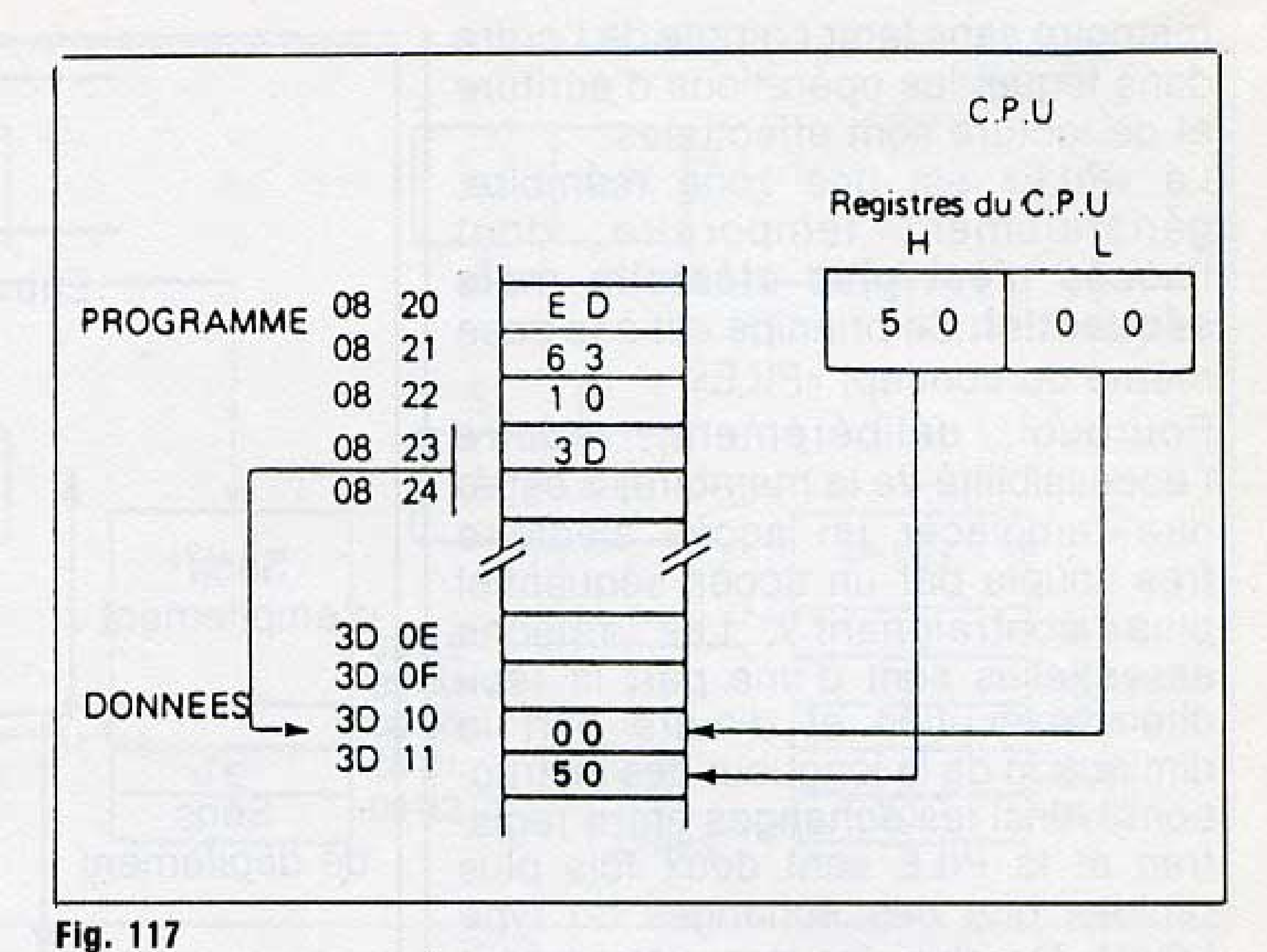


Fig. 117

Nous supposons que le programme commence à l'adresse 08 20 H.

La figure schématise le détail de fonctionnement de cette exécution.

III.9. Transferts entre registres 16 bits

Etant donné qu'il n'existe pas d'instructions de transfert à proprement parlé entre les registres 16 bits (comme celles du type LD r, r' pour les registres 1 octet), il faut utiliser deux fois l'instruction LD r, r'.

Le registre SP (Stack Pointer ou Pointeur de pile) fait partiellement exception à cette situation. En effet, il peut être chargé directement par le contenu de la paire de registres HL ou par le contenu de l'un des registres index IX ou IY.

Les mnémoniques et codes hexadécimaux sont :

LD SP, HL	F9
LD SP, IX	DD F9
LD SP, IY	FD F9

IV. LA PILE

IV.1. Définition

Les instructions de transfert que nous avons étudiées nous permettent de «stocker» ou d'«extraire» une donnée 8 ou 16 bits de la mémoire dans n'importe quel emplacement, l'accès à la mémoire est dit ALEATOIRE, et de plus l'adresse fait partie intégrale de l'opérande de l'instruction.

Nous pouvons encore dire plus simplement que les instructions du type LD (Load) permettent d'effectuer une suite d'échanges entre registres et n'importe quel emplacement

mémoire sans tenir compte de l'ordre dans lequel les opérations d'écriture et de lecture sont effectuées.

La «PILE» est une zone mémoire, généralement temporaire, dont l'**accès n'est plus aléatoire mais séquentiel.** Ce principe est à la base même du concept «PILE».

Pourquoi, délibérément, réduire l'accessibilité de la mémoire, c'est-à-dire remplacer un accès aléatoire très souple par un accès séquentiel plus contraignant ? Les raisons essentielles sont d'une part la rapidité d'exécution et d'autre part la diminution de la longueur des instructions. Ainsi les échanges entre registres et la PILE sont deux fois plus rapides que des échanges du type LOAD. De plus, les instructions sont d'un seul octet (deux pour les registres index) au lieu de trois ou quatre pour les instructions de transfert.

IV.2. L'intérêt de la pile

Le rôle essentiel de la PILE est de constituer une mémoire temporaire. Elle trouve tout particulièrement son utilité dans les opérations de sauvegarde des états du CPU et/ou de ses registres internes, dans le cas d'interruption de programme ou d'appel à des sous-routines.

Expliquons sommairement cette notion d'interruption qui sera reprise plus en détail par la suite. Rappelons tout d'abord qu'un microprocesseur ne peut effectuer, à un instant donné, qu'une seule tâche à la fois, cependant il est souvent amené à interrompre le programme principal pour exécuter une sous-routine ou programme secondaire.

Pour le bon déroulement du programme principal, le microprocesseur suspend momentanément le programme en cours, en ayant soin de sauvegarder dans la PILE les états du CPU et des registres, rendant ainsi l'ensemble disponible pour exécuter la sous-routine.

Cette opération terminée, les données de la pile sont restituées et le programme principal interrompu reprend son déroulement normal comme s'il n'y avait pas eu de «rupture».

Etant donné que les appels à des sous-routines sont fréquents pour obtenir une bonne efficacité, il est souhaitable que les opérations de sauvegarde et de restitution soient aussi rapides que possibles : c'est ce que permet justement la «PILE».

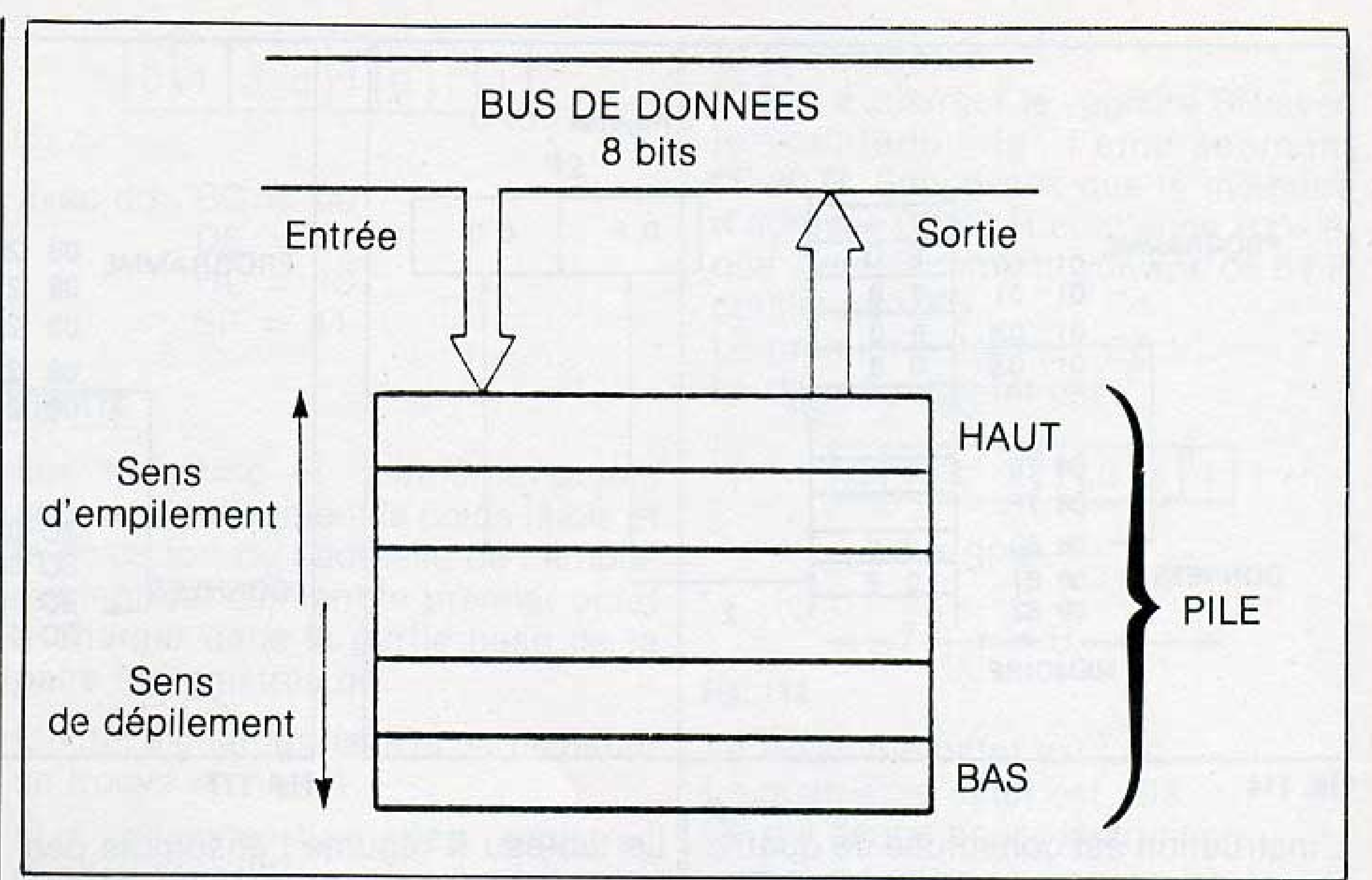


Fig. 118

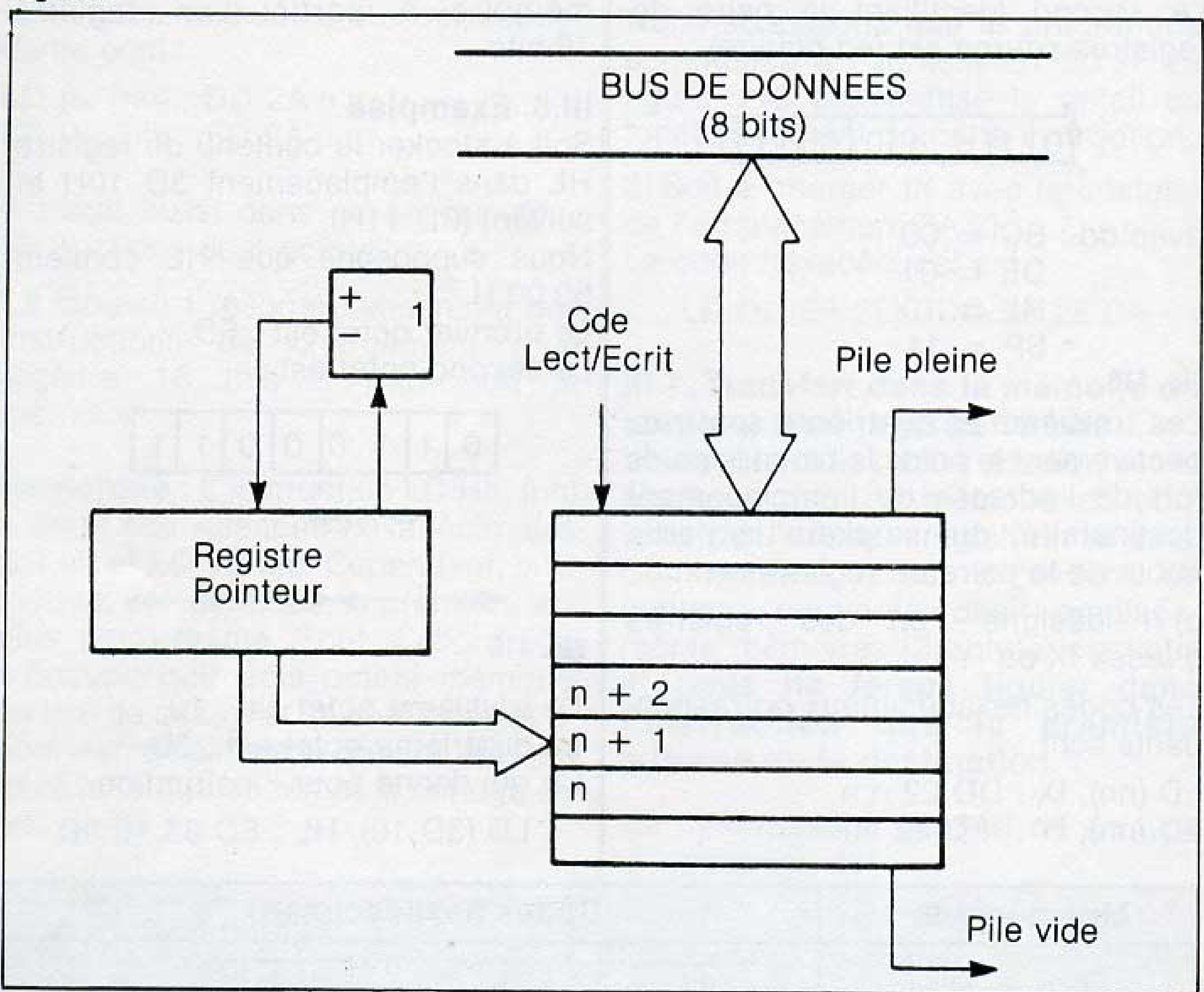


Fig. 119

IV.3. La pile hardware

La pile hardware est une mémoire spéciale, constituée d'un certain nombre de registres 8 bits par exemple, dont l'Accès Séquentiel s'effectue de la manière suivante : les informations introduites dans la mémoire **sont «empilées» les unes au-dessus des autres, en partant du bas vers le haut.** Inversement, elles **sont extraites du haut vers le bas.** On dit qu'il s'agit d'une mémoire «dernier entrée, premier sortie» ou LI-FO (Last In, First Out).

Il y a une certaine analogie avec une pile d'assiettes par exemple. La vaisselle faite, la ménagère range les assiettes (stockage) en les empilant les unes au dessus des autres, de bas en haut. Par contre, lorsqu'elle dresse la table, elle prend toujours l'assiette qui est au sommet de la pile (dépilement) : la dernière assiette rangée devient la première assiette qui retourne sur la table (LI-FO).

Examinons la figure 118 qui représente la «pile».

Les informations transitent par le bus de données et entrent dans la mémoire par le canal «Entrée». Elles retournent sur le bus par le canal «Sortie». Tout comme le «bus de données» qui est bidirectionnel, les deux canaux d'accès à la «pile» sont confondus en un seul bidirectionnel lui aussi (fig. 119).

Cependant, **l'absence de bus d'adresses doit surprendre le lecteur.** Celui-ci n'aura pas manqué de noter que nous avons parlé d'une **«Mémoire à Accès Séquentiel»** et non d'une «Mémoire à Accès Aléatoire».

Dans une mémoire à accès séquentiel, les données sont enregistrées les unes à la suite des autres (bande magnétique, par exemple) dans leur ordre d'arrivée, ou les unes au-dessus des autres (dans une pile par exemple) : **cette méthode exclut la lecture ou l'écriture d'un emplacement quelconque.**

Par contre, dans une mémoire à accès aléatoire, n'importe quel emplacement de la mémoire **est directement accessible** en le sélectionnant au moyen de son adresse.

Quelle que soit la réalisation hardware de la «pile», celle-ci possède un élément essentiel : un niveau (qu'on nomme pointeur» qui indique le «sommet» de la pile (figure 119) ou plus exactement l'emplacement du **«sommet + 1»,** c'est-à-dire le premier emplacement disponible.

Pour stocker une nouvelle donnée, celle-ci est inscrite dans l'emplacement ainsi pointé et le niveau augmente d'une unité.

Par contre, pour extraire une donnée, le niveau est d'abord baissé d'une unité, et le contenu de la case mémoire sélectionnée, est placé sur le bus de data.

La pile possède parfois deux autres indicateurs, l'un pour signaler que la **mémoire est pleine,** l'autre quand la **pile est vide.**

IV.4. La PILE SOFTWARE

Dans la plupart des systèmes, la pile est une portion de la zone mémoire vive disponible. L'indicateur de niveau est le registre «Pointeur de pile» ou «Stack Pointer» (SP), dont le contenu est toujours le premier emplacement disponible ou le sommet de la pile + 1.

Les opérations de traitement dans lesquelles la PILE intervient sont essentiellement des opérations de transfert sur 16 bits, dans lesquelles la destination ou la source est implicitement contenue dans le code opératoire lui-même.

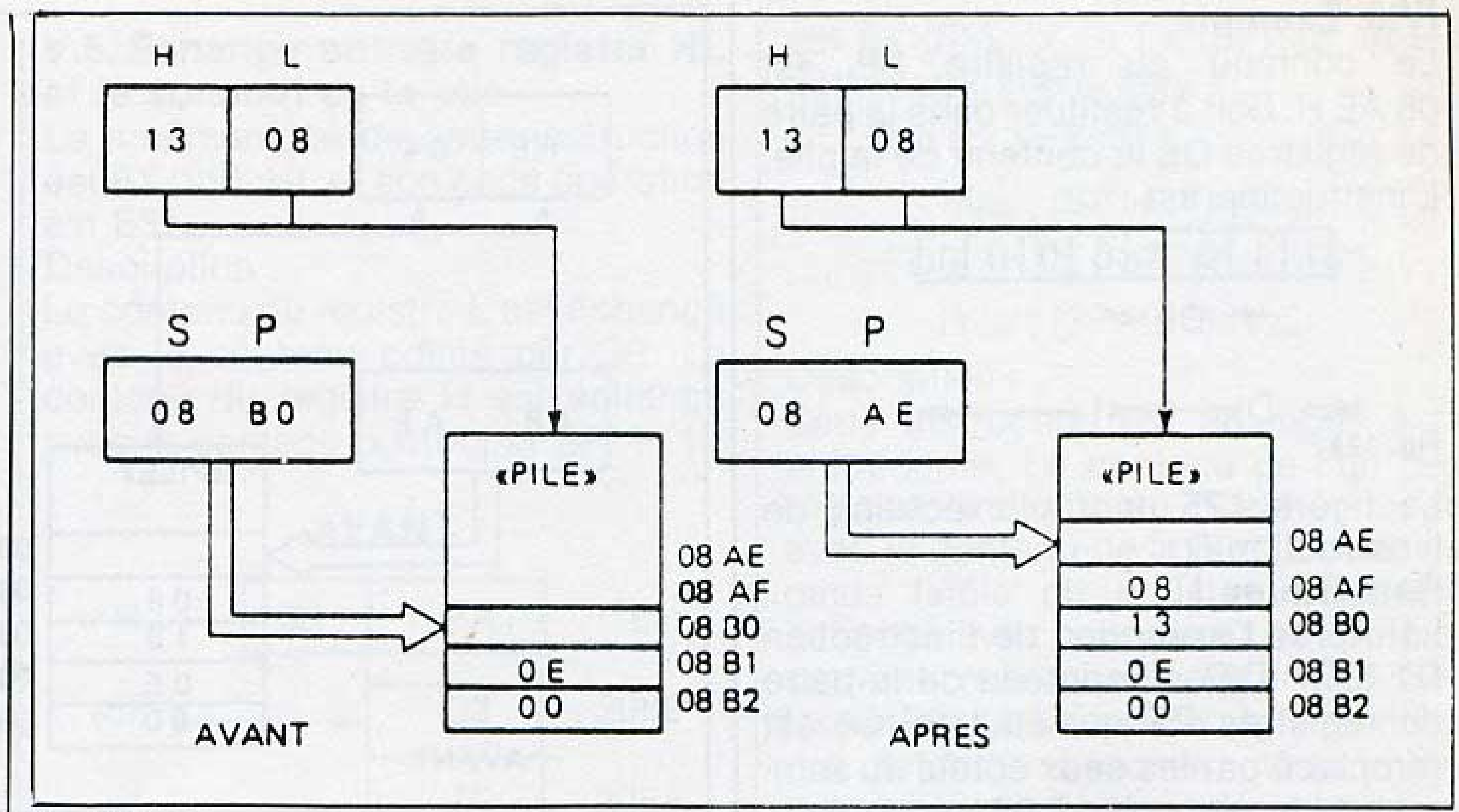


Fig. 122

La première opération est appelée «PUSHING» dont le mnémonique est PUSH : elle permet de garnir la pile avec le contenu de l'un des registres 16 bits (y compris AF).

La seconde opération est appelée «POPING» dont le mnémonique est POP : elle permet de dégarnir la pile, c'est-à-dire de transférer dans l'un des registres 16 bits, l'information contenue dans la pile.

IV.5. Instructions PUSH

a) Registres «Source» : AF, BC, DE ou HL

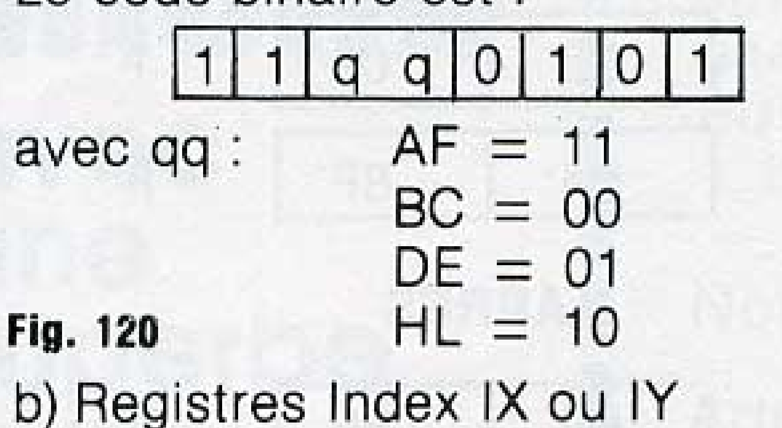
Le contenu (2 octets) de l'une des paires de registres AF, BC, DE ou HL est placé dans la pile.

Le schéma de l'instruction est :

Code opératoire	Source
Push	QQ
(pousser)	(paire de registres)

On notera l'absence de la destination qui est incluse implicitement dans le code opératoire (fig. 120).

Le code binaire est :



avec qq : AF = 11
BC = 00
DE = 01
HL = 10

Fig. 120

b) Registres Index IX ou IY

Le contenu (2 octets) du registre Index IX ou IY est placé dans la pile.

Les codes hexadécimaux sont :

Push IX : DD E5
Push IY : FD E5

IV.6. Exemple

Le contenu du registre SP est 08 B0 H. Soit à placer dans la pile le contenu de la paire de registres HL.

L'instruction est :

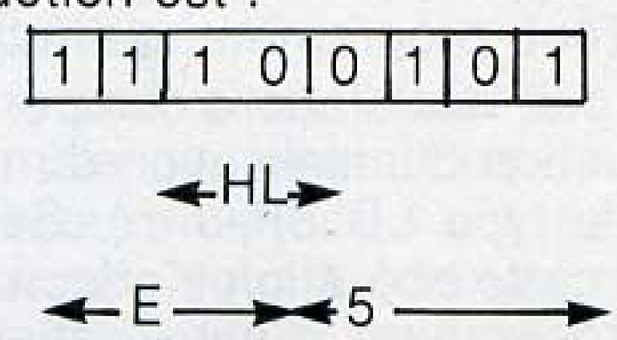


Fig. 121

La figure 122 décrit l'exécution de l'instruction E5.

IV.7. Instructions : Pop

a) Registres : AF, BC, DE ou HL

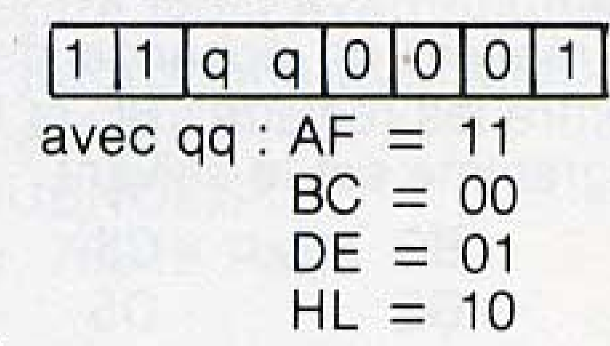
Les deux octets du sommet de la pile sont chargés dans l'une des paires de registres AF, BC, DE ou HL.

Le schéma de l'instruction est :

Code opératoire	Destination
Pop	qq
(Extraire)	(paire de registres)

On notera l'absence de la source (fig. 123).

Le code binaire est :



avec qq : AF = 11
BC = 00
DE = 01
HL = 10

Fig. 123

B) Registres Index IX ou IY :

Les deux octets du sommet de la pile sont chargés dans l'un des registres index IX ou IY.

Les codes hexadécimaux sont :

POP IX : DD E1
POP IY : FD E1

IV.8. Exemple

Le contenu du registre SP est 08 AE H. Soit à restituer dans la paire de registres DE le contenu de la pile. L'instruction est :

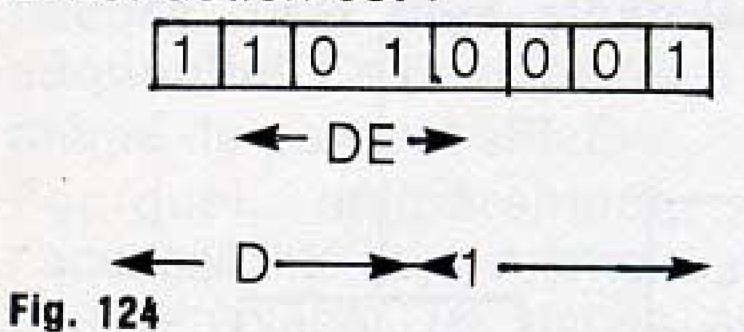


Fig. 124

La figure 125 décrit l'exécution de l'instruction D1.

Remarques I

I.1. Après l'exécution de l'instruction D1 (POP DE), le contenu de la paire de registres DE qui était FE 04 est remplacé par les deux octets du sommet de la pile soit 13 08.

I.2. Les deux octets qui viennent d'être chargés dans DE, sont toujours dans la PILE. Si nous effectuons un PUSH, les deux octets seront écrasés par cette nouvelle situation.

Remarques II

II.1. Tout programme qui utilise la PILE doit tout d'abord comporter une instruction d'initialisation du registre SP (du type LD SP, n n). Généralement cette opération s'effectue dans la sous-routine d'initialisation à la mise sous tension, c'est le cas dans le MPF-IB (voir le listing source, p. 2 ligne 116).

II.2. Le registre pointeur de Pile, étant un registre 16 bits, il peut donc adresser 64 Kbytes ; il est possible de construire la PILE dans n'importe quelle position de la mémoire, pourvue que celle-ci soit une zone «mémoire vive».

Remarques III

Le registre source dont le contenu est sauvegardé dans une opération de PUSH, peut ne pas être le même lors de la restitution (POP). Ceci permet de réaliser simplement des opérations de transfert entre registres 16 bits.

Exemple

Soit à réaliser un échange de données entre les registres BC et DE.
Le programme est le suivant

```
PUSH    BC    C5
PUSH    DE    D5
POP     BC    C1
POP     DE    D1
```

V. ECHANGES

V.1. Présentation

Dans la présentation hardware du CPU Z80, nous avons mentionné la présence d'un second ensemble de registres, identique au premier, dénommé «registres primes» (avec A', B', C', D', E', H' et L').

Les instructions de transfert étudiées jusqu'alors ne permettent pas d'utiliser ces registres auxiliaires.

En réalité, l'utilisateur **ne travaille qu'avec un seul jeu à la fois,** mais il arrive souvent, pour répondre à une demande d'interruption par exemple, que un ou plusieurs registres doivent être libérés rapidement tout en conservant leur contenu.

Ceci est rendu possible grâce aux instructions d'échanges. De plus, leur temps d'exécution, pour les instructions de 1 octet, est très court (4 *ms* avec l'horloge à 1 MHz).

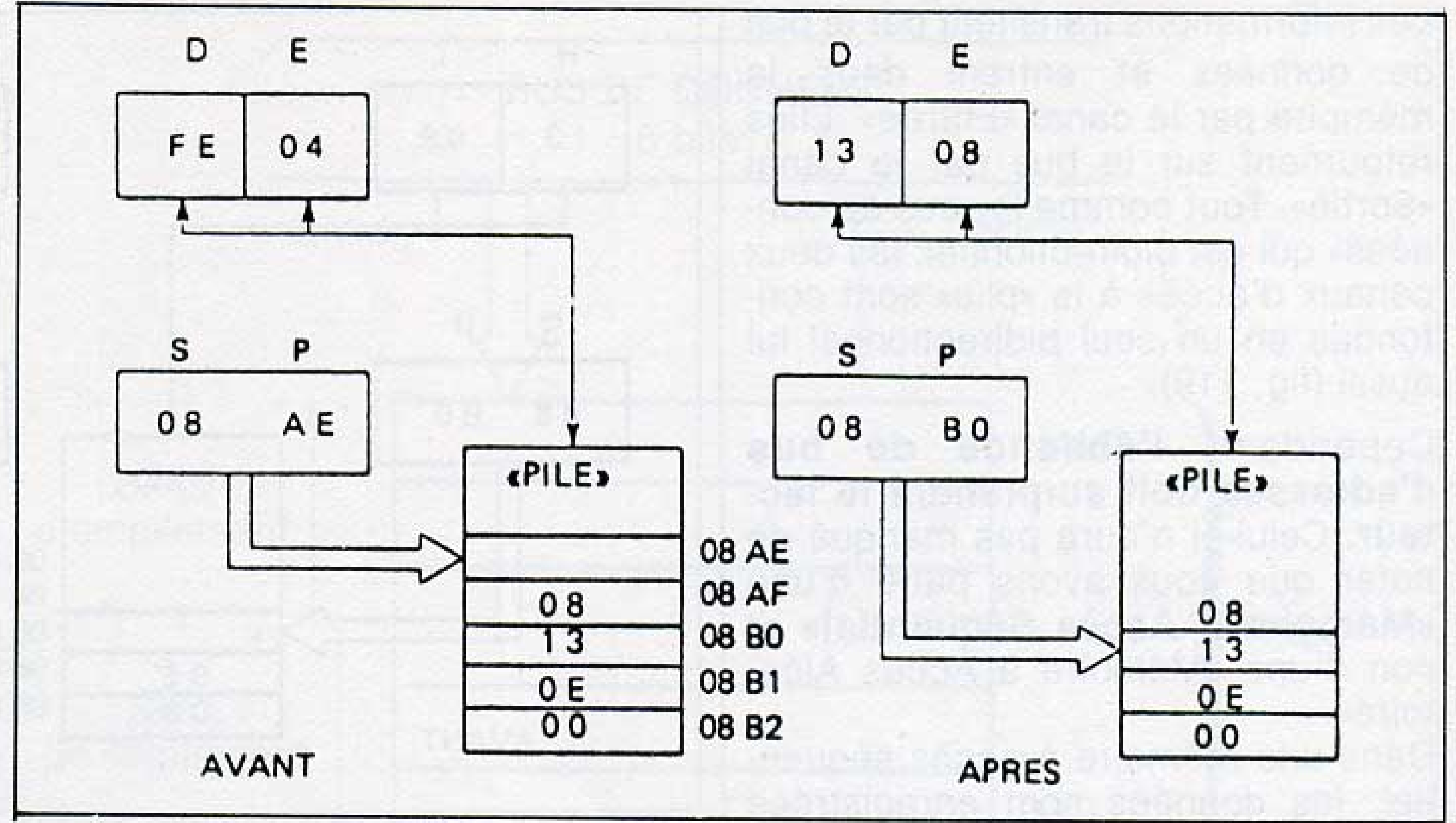


Fig. 125

V.2. Echange entre les registres AF et A'F'

Le mnémonique de cette instruction est EX AF, A'F' et son code opération : 08.

Description :

Le contenu des registres A et F sont respectivement échangés avec le contenu des registres A' et F'.

AVANT

A	73	09	F
A'	B4	8F	F'

APRES

A	B4	8F	A'
A'	73	09	F'

Notons que deux fois (ou un nombre pair) l'exécution de l'instruction EX ramène le système dans son état initial.

V.3. Echange entre les registres auxiliaires

Le mnémonique de cette instruction est EXX et son code opération : D9.

Description :

Les contenus des registres BC, DE et HL sont respectivement échangés avec les contenus de B'C', D'E' et H'L'.

AVANT

A	73	09	F	Inchangé
B	2A	27	C	
D	CD	F4	E	Echangé
H	30	FA	L	

APRES

A	73	09	F
B	D3	01	C
D	06	2B	E
H	21	03	L

AVANT

A'	B4	8F	F'
B'	D3	01	C'
D'	06	2B	E'
H'	21	03	L'

APRES

A'	B4	8F	F' Inchangé
B'	2A	27	C' Echangé
D'	CD	F4	E' Echangé
H'	30	FA	L' Echangé

V.4. Echange entre les registres DE et HL

Le mnémonique de cette instruction est EX DE, HL et son code opération est EB.

Description :
Les contenus des paires de registres DE et HL sont échangés entre eux.

AVANT

D	CD	F4	E
H	30	FA	L

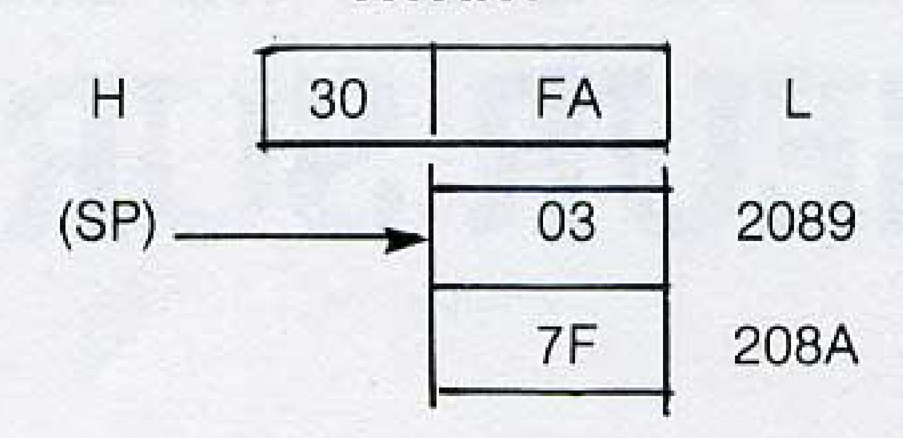
APRES

D	30	FA	E
H	CD	F4	L

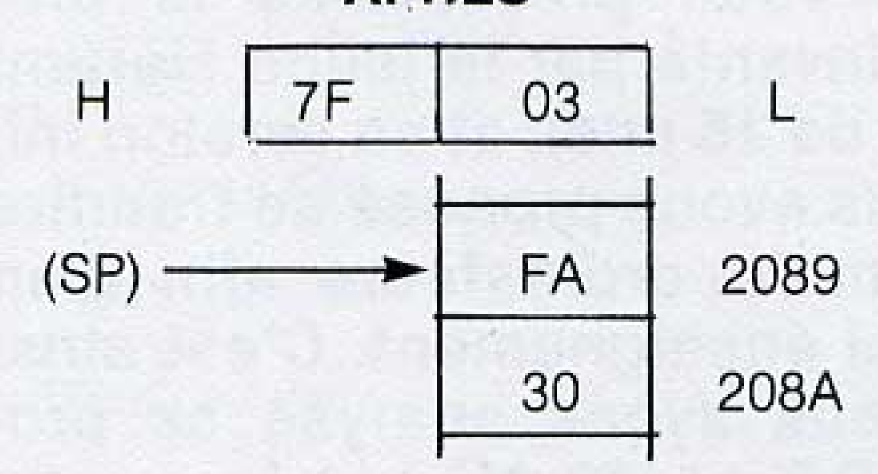
V.5. Echange entre le registre HL et le sommet de la pile

Le mnémonique de cette instruction est EX (SP), HL et son code opération est E3.
Description :
Le contenu du registre L est échangé avec le contenu pointé par SP. Le contenu du registre H est échangé avec le contenu pointé par SP + 1.

AVANT



APRES



V.6. Echanges entre les registres IX ou IY et le sommet de la pile

Le mnémonique de cette instruction est EX (SP), IX ou EX (SP), IY, et les codes opérations sont :

DD E3 : EX (SP), IX ou (SP) IX_{bas} ; (SP + 1) IX_{haut}

FD E3 : EX (SP), IY ou (SP) IY_{bas} ; (SP + 1) IY_{haut}

Description :
Cette instruction est analogue à la précédente, Le contenu de l'un des registres index IX ou IY est échangé avec le contenu de la pile. L'octet de poids faible du registre index est échangé avec le contenu de l'emplacement mémoire pointé par SP. L'octet de poids fort du registre index est échangé avec le contenu de l'emplacement mémoire pointé par (SP) + 1.
A noter que le temps d'exécution est de 22 micro-secondes (avec horloge à 1 MHz), tandis que la précédente, échange entre HL et le sommet de la pile, est de 19 micro-secondes.

Conclusion

Nous avons ainsi passé en revue l'ensemble des opérations de transferts : mouvement de 1 octet puis de deux octets. Nous étudierons dans le prochain numéro les opérations «arithmétiques» et «logiques».

Philippe Duquesne

COURS PRATIQUE DE MICROPROCESSEUR AVEC LE MICROPROFESSOR MPF-IB

SEPTIEME PARTIE

Le langage du Z80R (3) (suite)

SOMMAIRE

I. INTRODUCTION

Poursuivant notre étude du langage du microprocesseur, nous allons aborder dans cette septième partie, les opérateurs logiques et arithmétiques.

Les leçons 5 et 6 étaient consacrées aux mouvements de données internes dans le cas de transfert entre registres mouvements externes quand la donnée était le contenu d'un emplacement mémoire. Nous pouvons dire qu'il s'agissait d'«opérateurs neutres» dans la mesure où les **données ne sont pas modifiées mais transférées.**

Dans les instructions que nous allons étudier dès à présent, le résultat que nous obtiendrons est fonction d'une part de l'opérateur désigné mais aussi des données, celles-ci pouvant résulter d'un transfert.

Le résultat, lui aussi, peut être transféré dans un emplacement mémoire pour y être sauvegardé : ainsi opérateurs de transfert et opérateurs logiques et arithmétiques sont complémentaires pour obtenir le bon déroulement d'un programme.

Pour que tout soit bien clair, au risque «d'enfoncer une porte ouverte», nous rappelons brièvement les opérateurs de base dans le cadre «microprocesseur».

II. OPERATIONS DE BASE

II.1. Définition

Considérons le schéma électrique représenté par la figure 126.

Il comprend, montés en série :

— 1 générateur G
— 1 interrupteur I
— 1 poussoir P
— 1 lampe L

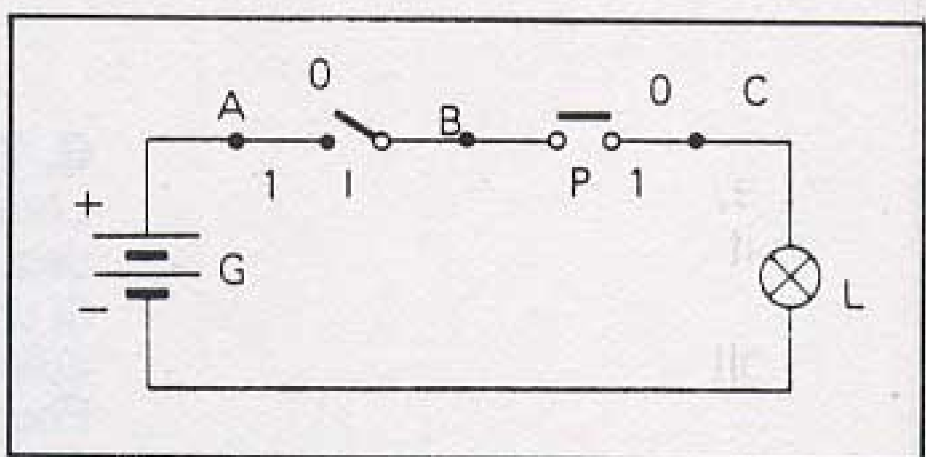


Fig. 126

L'interrupteur I peut prendre deux positions. Quand il est «OUVERT» (ou inactif) le courant ne peut circuler de A vers B. Par contre, lorsque I est «FERME» (ou actif) le courant peut circuler.

Le poussoir P peut prendre deux positions.

Quand P est :

— relâché (inactif), le courant ne peut pas circuler de B vers C

— enfoncé (actif), le courant peut circuler de B vers C.

La lampe L est soit «éteinte» soit «allumée».

L'interrupteur I et le poussoir P sont des variables logiques telles que nous les avons définies. En effet I est soit ouvert ou fermé ; P est soit relâché ou enfoncé ; la lampe est soit allumée ou éteinte.

Les trois variables I, P et L ne jouent pas le même rôle. L'état de la lampe (éteinte ou allumée) dépend de l'état des deux autres variables I et P. La lampe L sera allumée si l'interrupteur I ET le poussoir P sont l'un ET l'autre dans l'état actif.

Nous dirons que l'état de la varia-

ble de sortie L dépend de l'état des deux autres variables d'entrée I et P.
L'ensemble I et P, ainsi monté, **réalise la fonction logique ET,** c'est-à-dire que la lampe L sera allumée (état 1) quand **l'une et l'autre des variables «I» ET «P» sont simultanément dans l'état actif (état 1).**
Donc un «interrupteur» et un «poussoir» en série constituent un OPERATEUR qui réalise **la fonction ET.**
Modifions le circuit comme l'indique la figure 127.

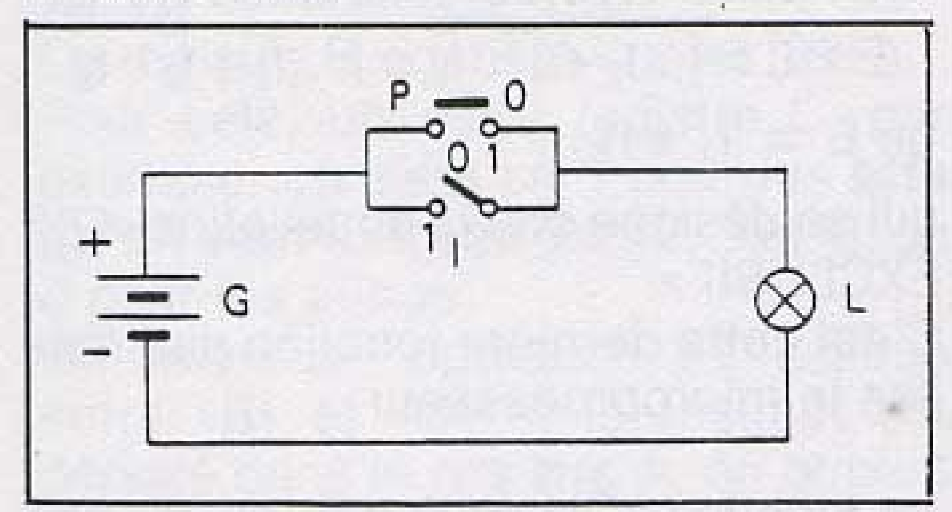


Fig. 127

Les deux commandes «I» et «P» sont en parallèle au lieu d'être en série.
Le montage de I et P réalise un autre opérateur. Pour que la lampe L soit allumée, il faut que :
— I soit fermé
OU
— P soit enfoncé.
Nous dirons que cet ensemble réalise **la fonction logique «OU».**
Examinons maintenant la figure 128.

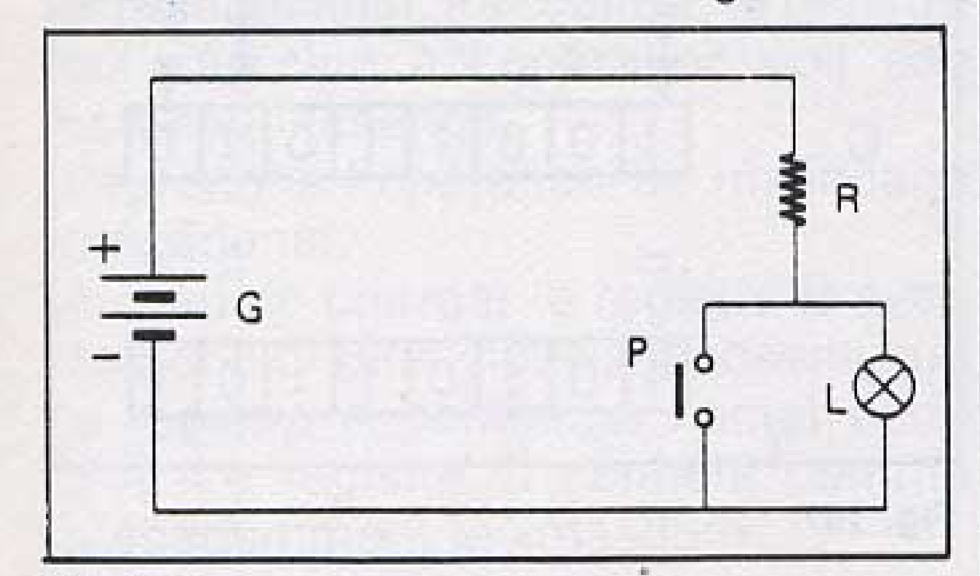


Fig. 128

Quand P est relâché (état 0), la lampe L est allumée (état 1).
Quand P est enfoncé (état 1), la lampe L est éteinte (état 0).
Il s'agit d'un autre opérateur puisque les états de la variable d'entrée et de la variable de sortie sont l'INVERSE l'un de l'autre. L'opérateur qui réalise cette fonction est un **INVERSEUR.**

II.2. Opérateur «ET»

Reprenons le schéma de la figure 126 et par convention nous désignerons, mais arbitrairement, par «1» les états actifs des variables et par conséquent par «0» les états inactifs.
A l'aide de ces notations, nous pouvons établir les deux tableaux des figures 129 et 129 bis. Tous deux donnent l'état de la variable de sortie (la lampe L) en fonction des positions des variables I et P.

Inter I	Poussoir P	Lampe L
Pos. 0	Relâché	Eteinte
Pos. 1	Relâché	Eteinte
Pos. 0	Enfoncé	Eteinte
Pos. 1	Enfoncé	Allumée

Fig. 129

I	P	L
0	0	0
1	0	0
0	1	0
1	1	1

Fig. 129 bis

Ces deux tableaux nous permettent de résumer la fonction réalisée par ce circuit de la manière suivante : **la lampe L est allumée quand l'interrupteur I ET le poussoir P sont, l'UN ET l'AUTRE dans l'état actif.**
Ce qui peut se traduire par l'équation logique :

$$L = I \times P$$

Le signe «X» (qu'il ne faut pas confondre avec le signe «multiplié») est celui de la fonction ET.
Que se passe-t-il dans un microprocesseur ?
La fonction ET entre le contenu de deux registres B et C par exemple, consiste à effectuer «bit à bit» l'opération ET entre chaque bit de même position des deux registres. Le résultat est généralement déposé dans l'accumulateur (figure 130).

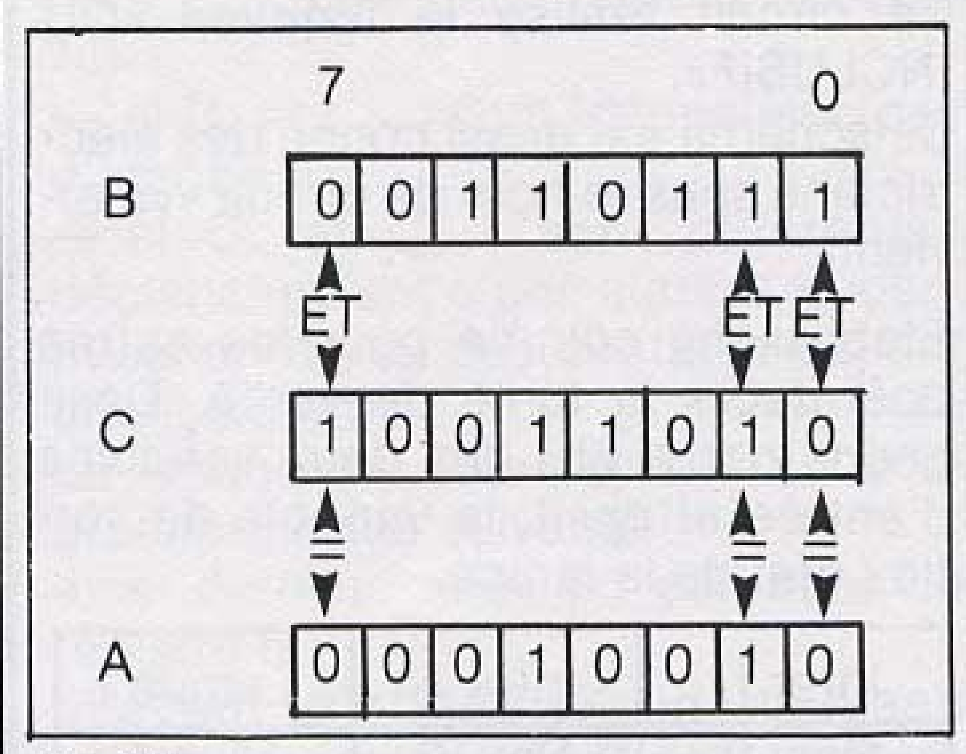


Fig. 130

Ainsi :

$$A_0 = B_0 \times C_0$$

ou 1 × 0 = 0

$$A_1 = B_1 \times C_1$$

ou 1 × 1 = 1

....

$$A_7 = B_7 \times C_7$$

ou 0 × 1 = 0.

II.3. Opérateur «OU»

D'une manière analogue, nous pouvons établir les deux tableaux des figures 131 et 131 bis.
Tous deux donnent l'état de la variable de sortie en fonction des positions des variables I et P dans le cas d'un opérateur OU.

Inter I	Poussoir P	Lampe L
Pos. 0	Relâché	Eteinte
Pos. 1	Relâché	Allumée
Pos. 0	Enfoncé	Allumée
Pos. 1	Enfoncé	Allumée

Fig. 131

I	P	L
0	0	0
1	0	1
0	1	1
1	1	1

Fig. 131 bis

La lampe L est allumée quand l'interrupteur I OU le poussoir P est dans l'état actif.
Ce qui peut se traduire par l'équation logique :

$$L = I + P$$

Le signe « + » (qu'il ne faut pas confondre avec le signe addition) est celui de la fonction OU.
Etudions à nouveau ce qui se passe dans un microprocesseur (fig. 132).
La fonction OU entre les contenus de deux registres B et C, consiste à effectuer «bit à bit» l'opération OU entre chaque **bit de même rang** des deux registres. Le résultat est généralement déposé dans l'accumulateur.

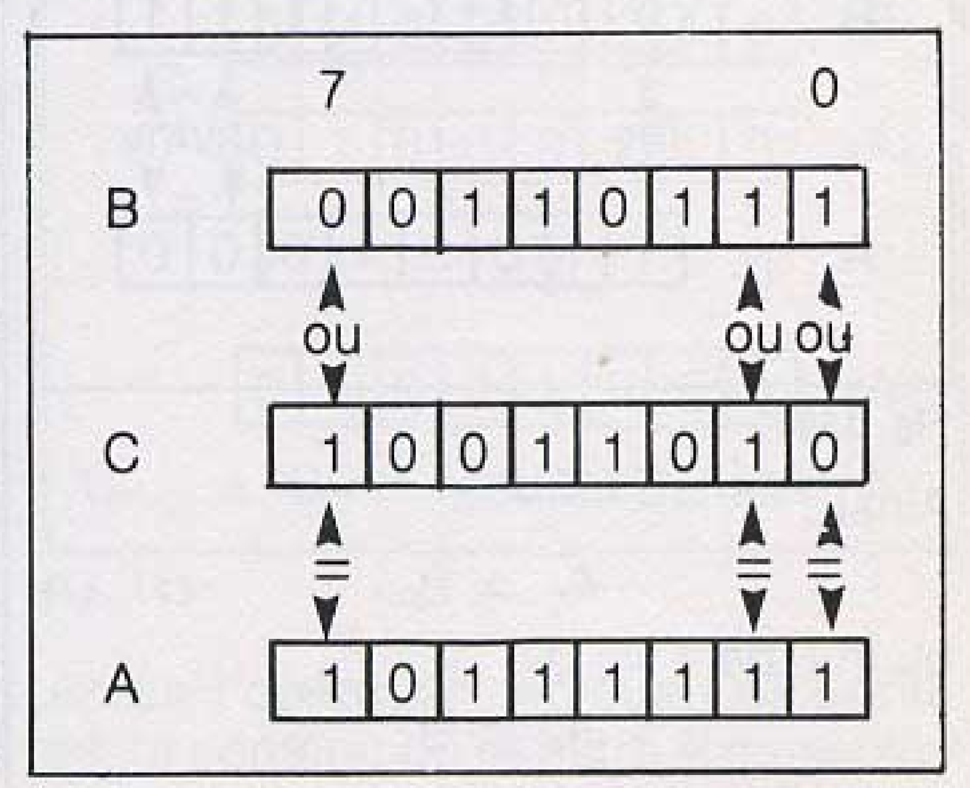


Fig. 132

Ainsi :

$$A_0 = B_0 + C_0$$

ou 1 + 0 = 1

$$A_1 = B_1 + C_1$$

ou 1 + 1 = 1

....

$$A_7 = B_7 + C_7$$

ou 0 + 1 = 1

II.4. Opérateur «inverseur»

Dans le cas de cet opérateur dit «inverseur» ou complément à 1, le tableau est réduit comme l'indiquent les figures 133 et 133 bis. Dans ce cas, l'état de la variable de sortie est inverse de celui de la commande.

Poussoir P	Lampe L
Relâché	Allumée
Enfoncé	Eteinte

Fig. 133

P	L
0	1
1	0

Fig. 133 bis

La lampe L est allumée quand le poussoir est relâché (inactif) et inversement, éteinte quand le poussoir est enfoncé (actif), ce qui peut se traduire par l'équation logique :

$L = \overline{P}$ où « ¯ » indique l'état inverse de la variable.

Au niveau du microprocesseur, l'exécution d'une telle instruction consiste à inverser chaque bit du registre considéré et de déposer le résultat dans A.

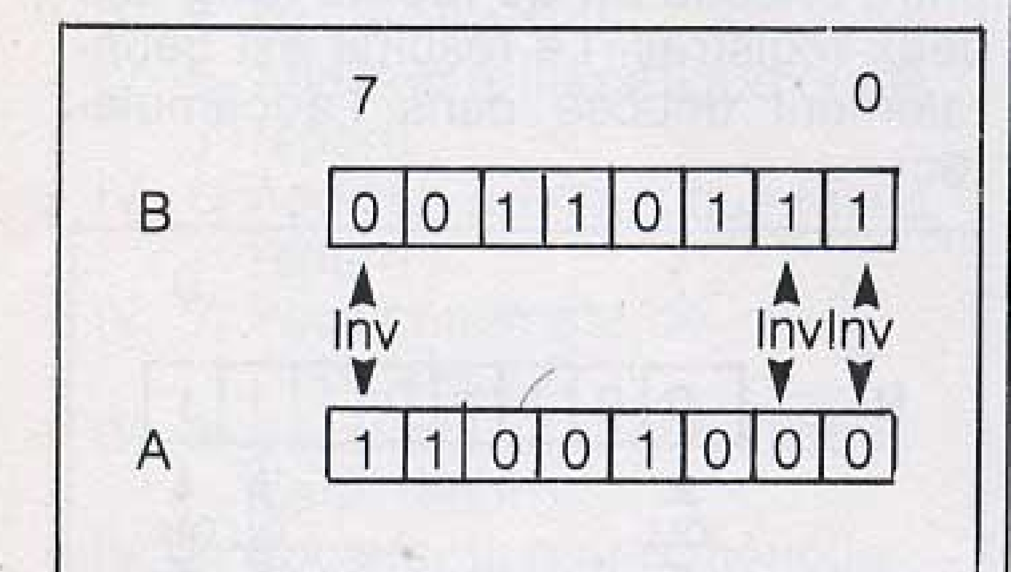


Fig. 134

Ainsi :

$$A_0 = \overline{B_0}$$

ou inv. de 1 = 0

$$A_1 = \overline{B_1}$$

ou inv. de 1 = 0

....

$$A_7 = \overline{B_7}$$

ou inv. de 0 = 1.

II.5. Opérateur «OU INCLUSIF»

Considérons le schéma de la figure 135. Le circuit comporte :

— un générateur G
— deux interrupteurs I_1 et I_2
— une lampe L.

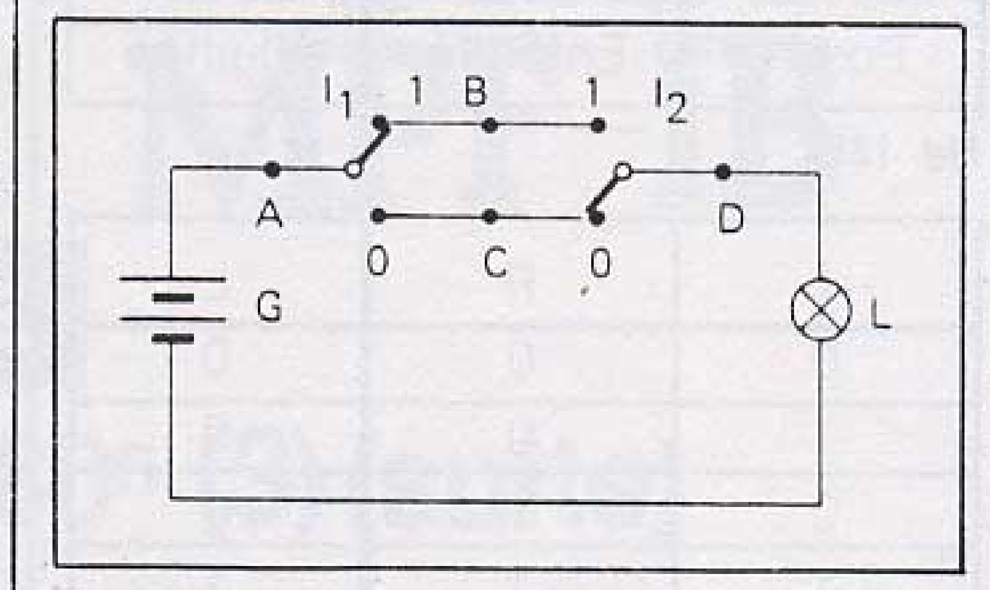


Fig. 135

Lorsque les interrupteurs I_1 et I_2 sont dans la position indiquée sur la figure (I_1 en 1, I_2 en 0), le courant ne circule pas : la lampe est éteinte.

Plaçons I_2 en position «1» sans changer I_1, le courant circule au travers de la branche ABD.

Si maintenant, au lieu de modifier I_2, nous avions placé I_1 en position «0» (I_2 étant resté en «0»), le courant circulerait au travers de la branche ACD.

En conclusion, **si les interrupteurs I_1 et I_2 sont tous les deux dans la position «1» (branche ABD) ou tous les deux dans la position «0» (branche ACD), la lampe L est allumée (ou position «1»).**

Par contre, si les variables I_1 et I_2 sont dans des états inverses «1» et «0» ou «0» et «1»), la lampe L est éteinte (ou position «0»).

Ce circuit réalise la fonction «OU INCLUSIF».

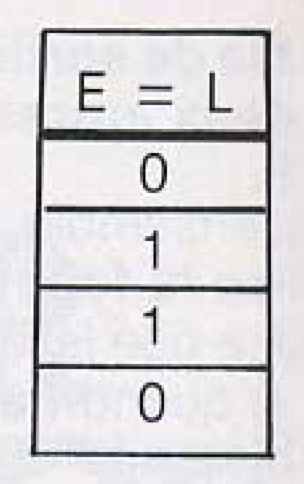
Ce schéma est aussi connu des électriciens sous le nom de circuit «va-et-vient».

Etablissons comme pour les autres opérateurs la table de vérité. Désignons par I_1 et I_2 les deux variables d'entrée et par L la variable de sortie : état de la lampe.

Inter I_1	Inter I_2	Lampe L
Pos. 0	Pos. 0	Allumée
Pos. 1	Pos. 0	Eteinte
Pos. 0	Pos. 1	Eteinte
Pos. 1	Pos. 1	Allumée

Fig. 136

I_1	I_2	L
0	0	1
1	0	0
0	1	0
1	1	1



E = L
0
1
1
0

Fig. 136

ce qui se traduit par l'équation logique :

$$L = I_1 . I_2 + \overline{I_1} . \overline{I_2}$$

En réalité, on lui préfère bien souvent l'opération inverse, telle que :

$$E = \overline{L} = \overline{I_1 . I_2 + \overline{I_1} . \overline{I_2}} = I_1 . \overline{I_2} + \overline{I_1} . I_2$$

ou $E = I_1 \oplus I_2$

qui se désigne sous l'appellation «OU EXCLUSIF».

C'est cette dernière fonction que réalise le microprocesseur.

La fonction «OU exclusif» (ou «XOR») entre deux registres B et C par exemple, consiste à effectuer bit à bit l'opération OU exclusif entre chaque bit de même rang des deux registres.

Le résultat est généralement déposé dans l'accumulateur.

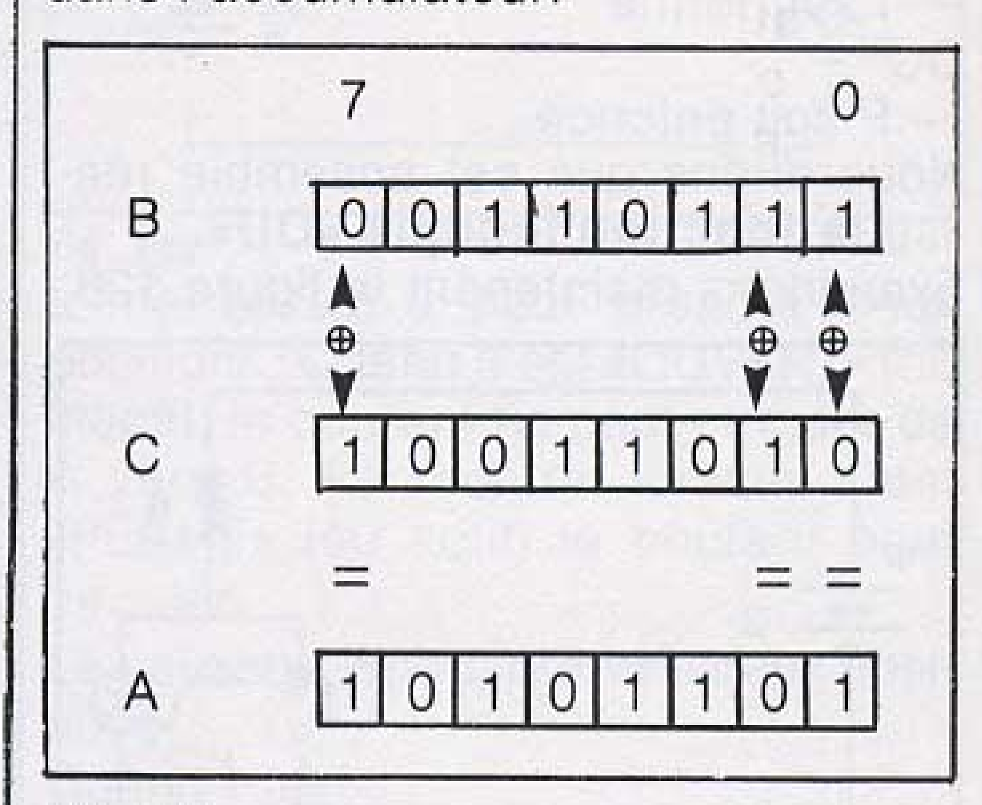


Fig. 137

Ainsi :

$$A_0 = B_0 \oplus C_0$$

ou 1 ⊕ 0 = 1

$$A_1 = B_1 \oplus C_1$$

ou 1 ⊕ 1 = 0

....

$$A_7 = B_7 \oplus C_7$$

ou 0 ⊕ 1 = 1.

II.6. Exemple 1 : Opérateur «ET»

Nous allons montrer comment «isoler» un certain nombre de digits inclus dans un octet. Pour fixer les idées, on a besoin d'isoler les bits 2, 3 et 4 du registre B qui contient 0011 0111 (ou 37H).

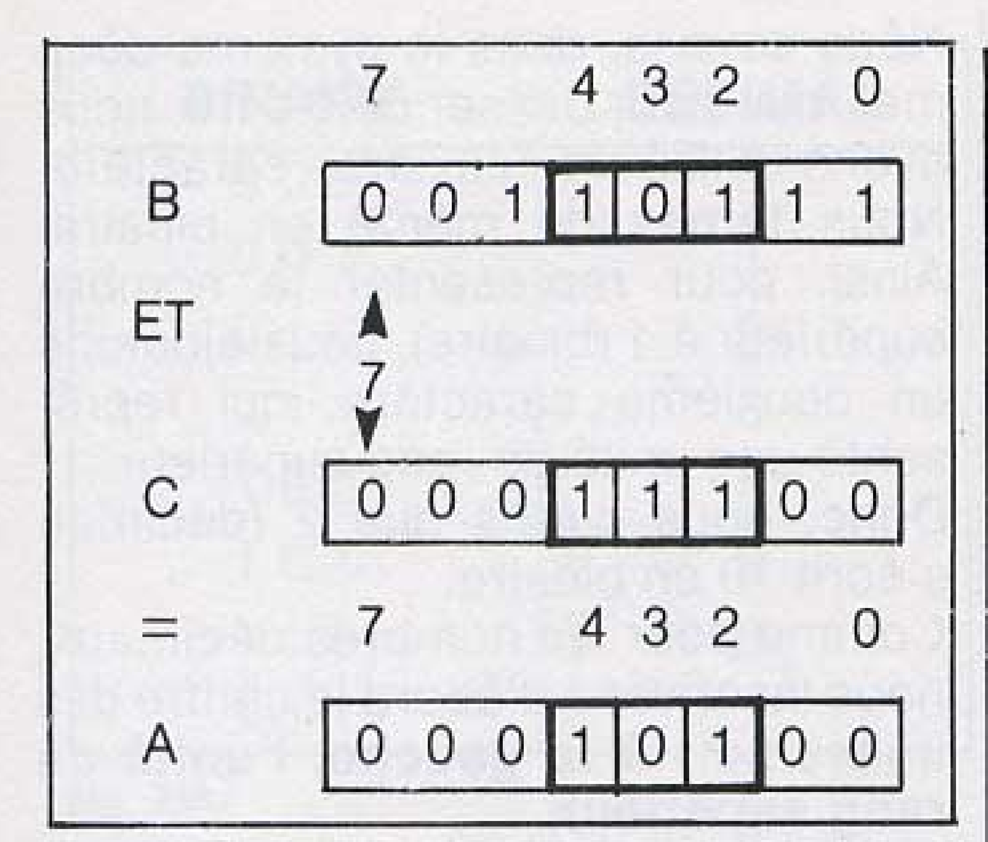


Fig. 138

Le registre B contient l'octet (37H). Pour cela, un autre registre C, par exemple, contient un 1 pour les **bits qu'on veut extraire** (bits 2, 3 et 4), un 0 pour les autres.

On effectue l'opération ET logique entre «B» et «C» et le résultat est déposé dans le registre A, on obtient ainsi les trois bits qu'on devait isoler (figure 138).

En réalité, dans les opérations logiques (et d'autres), le registre A joue un double rôle : il contient **d'une part l'une des données et constitue aussi le registre destinataire de l'opération.** Nous verrons ultérieurement comment cela est possible. Dans ce cas, la donnée qui était dans le registre A est détruite, puisque le contenu de celui-ci est remplacé par le résultat final. Par contre, le registre qui participe à l'opération voit son contenu inchangé.

Reprenons l'opération de masquage précédente.

Au lieu de charger le registre B avec l'octet d'origine, nous supposons que le registre A contienne l'octet complet. Le registre C contient comme précédemment le «masque».

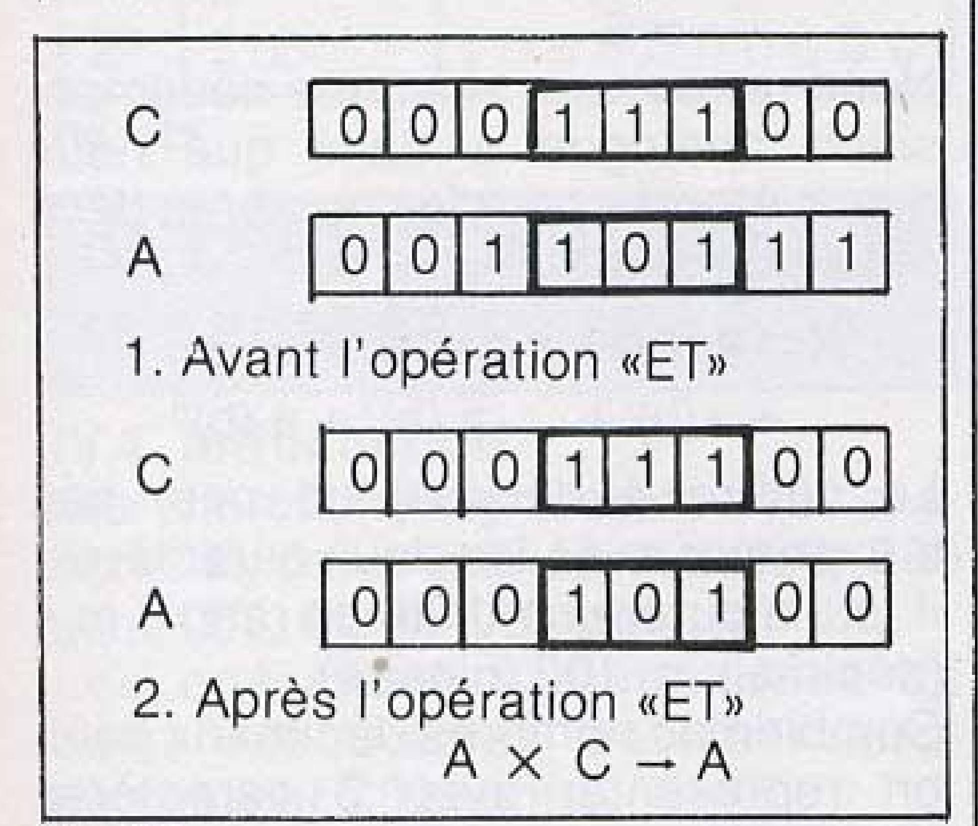


Fig. 139

Nota : X indique que l'opération est un ET.

Notons qu'il suffit :

— de charger le registre C avec le «masque»

— de charger le registre A avec l'octet d'origine

— d'effectuer l'opération ET logique entre A et C, il en résulte que :

• le contenu du registre A est la partie isolée

• le contenu du registre C est inchangé.

II.7. Exemple 2 : Opérateur «OU exclusif»

A l'aide d'un second exemple, nous allons montrer une application de la fonction «OU exclusif».

Soit un dispositif de sécurité ou d'alarme, qui comporte 8 contacts à surveiller. (On peut imaginer qu'il s'agit d'un dispositif de surveillance dans une maison ou un appartement).

Ceux-ci peuvent être soit ouverts (état «0») ou fermés (état «1»).

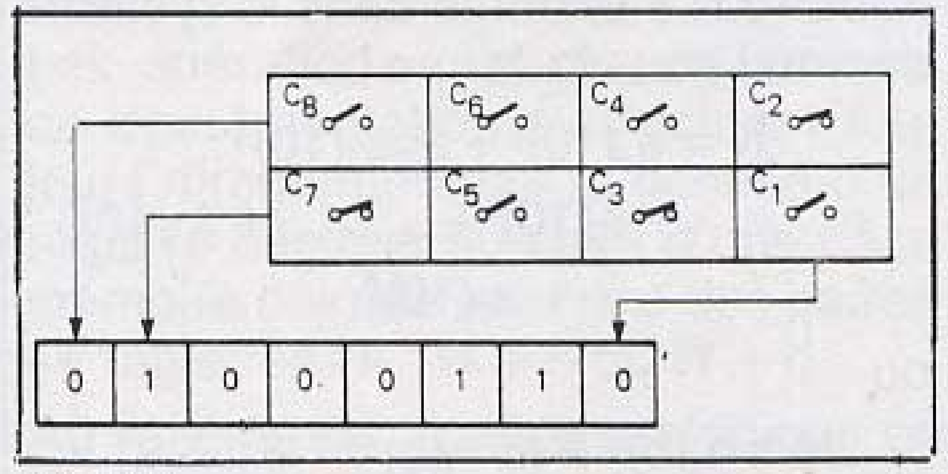


Fig. 140

Lorsque le système d'alarme est en fonctionnement, tout changement de l'état 1 ou plusieurs contacts, doit être détecté et les contacts identifiés.

Le rôle du microprocesseur est de «surveiller les contacts», d'effectuer périodiquement un relevé «état des contacts». Le système génère ainsi un mot de contrôle que nous désignerons par C_{n+1} et le compare au précédent C_n. Entre deux relevés successifs :

— si C_n et C_{n+1} sont identiques : pas d'alarme

— si C_n et C_{n+1} sont différents : déclenchement d'une alarme et identification du ou des contacts qui ont changé d'état.

La séquence (ou portion du programme) est la suivante :

a) le dernier octet C_n est dans le registre B

b) avant d'effectuer un nouveau relevé, le contenu de B est transféré dans A

c) le nouveau relevé C_{n+1} est chargé dans le registre B

d) l'opération A ⊕ B est effectuée, le résultat est déposé dans l'accumulateur A.

a) Sauvegarde de C_{n+1}

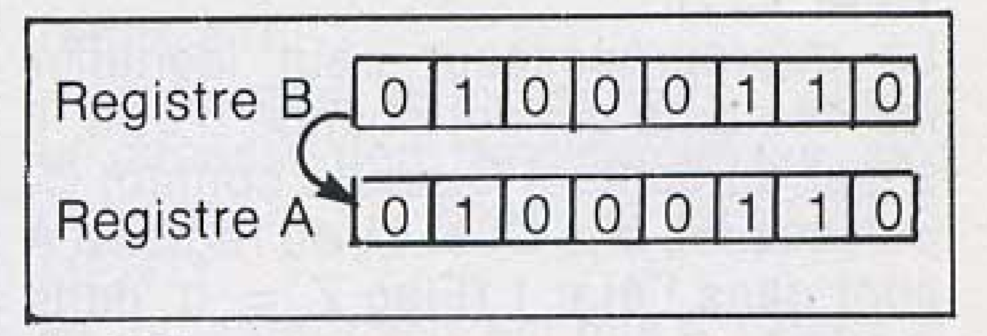


Fig. 141

b) Comparaison entre C_n et C_{n+1}

A partir de là, deux cas peuvent se présenter, comme l'illustrent les figures 142 et 143 ci-après.

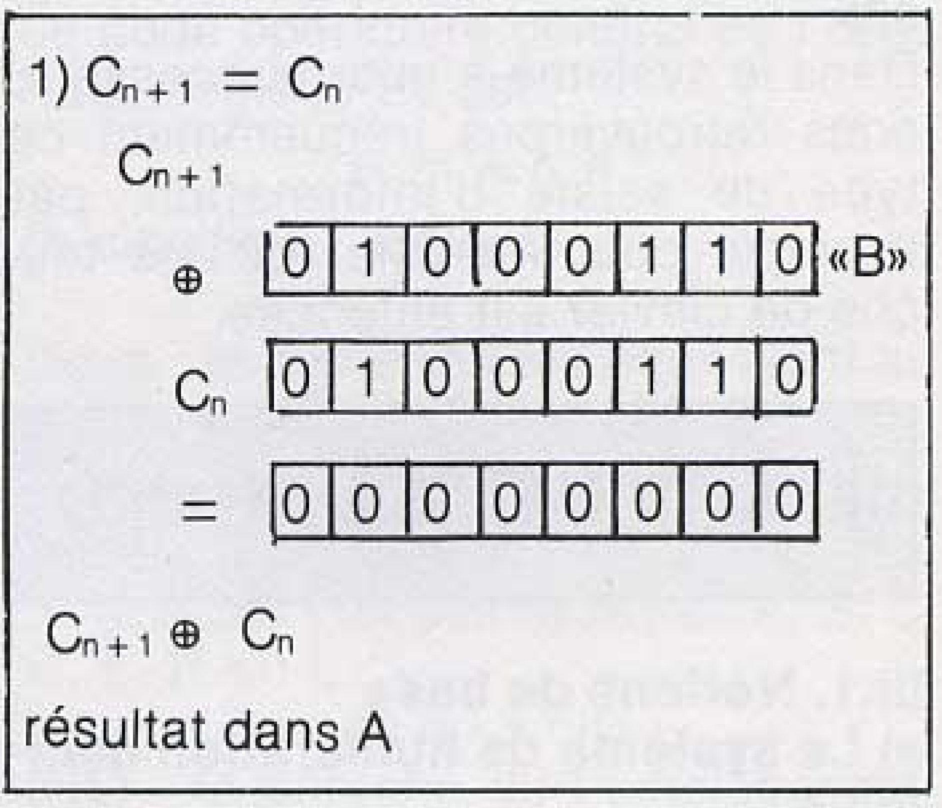


Fig. 142

Rappelons qu'avant l'opération logique «OU exclusif», le contenu du registre A était C_n, et celui du registre B : C_{n+1}. Après l'opération logique C_n est perdu, le contenu de A est le résultat $C_{n+1} \oplus C_n$. Le contenu du registre B est C_{n+1}, inchangé.

Dans ce cas, étant donné que $C_{n+1} = C_n$, le résultat de l'opération est une suite de 8 zéros : il n'y a eu aucun changement d'état entre les instants T_n et T_{n+1}. Pas d'alarme.

Nous verrons ultérieurement qu'une suite de 8 zéros dans le registre A résultant d'une opération logique (ou arithmétique) se détecte facilement (Flag Z en 1).

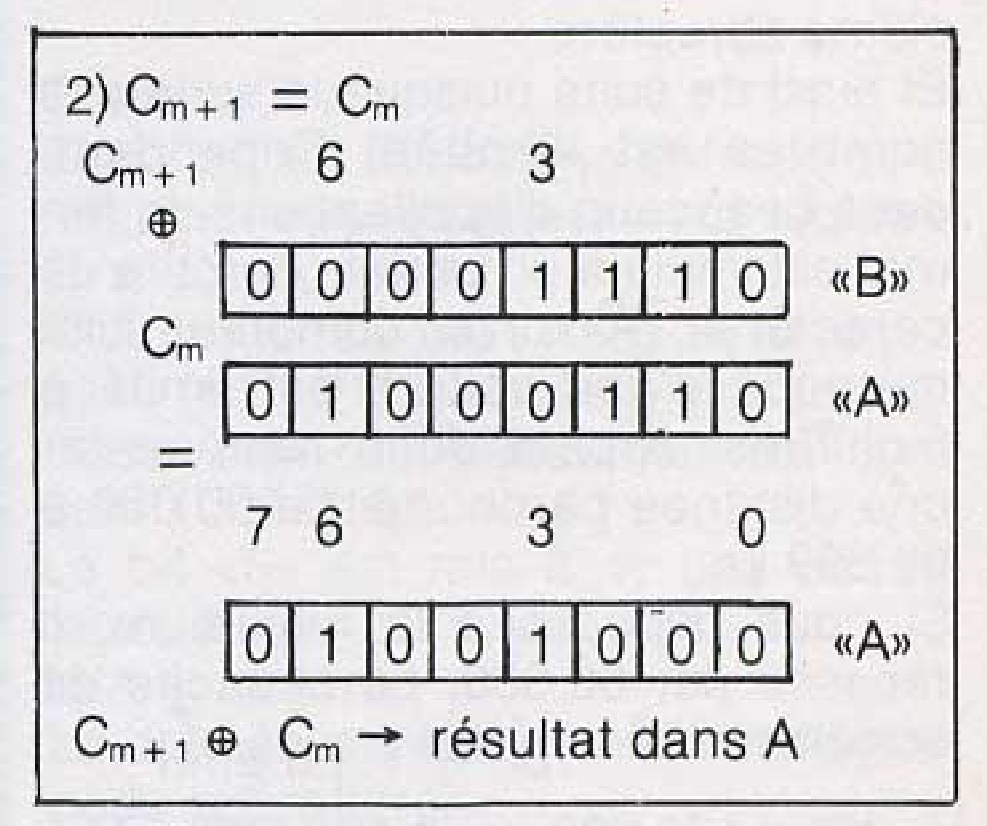


Fig. 143

Après l'opération logique OU exclusif, le contenu du registre A n'est plus «nul» : donc il y a **au moins un des**

contacts qui a changé d'état entre T_m et T_{m+1}.
Le microprocesseur peut identifier les contacts qui ont changé d'état, il s'agit du contact C_4 et du contact C_7 puisque les bits C_3 et C_6 du registre A sont dans l'état 1 (Flag Z = 0, dans ce cas).
En conclusion, la fonction «OU exclusif» a été mise à profit pour détecter d'une part et identifier d'autre part le ou les contacts qui ont changé d'états entre deux relevés successifs.
Dans le système à microprocesseur, nous retrouverons fréquemment ce type de saisie d'information, par exemple pour identifier qu'une touche de clavier est enfoncée.

III. ARITHMETIQUE BINAIRE

III.1. Notions de base :

a) Le système de numération décimale :

Notre façon de compter, le système de numération décimale nous est très familier. Nous en rappelons cependant les principes essentiels pour définir la notion de «base».
Notre système de numération décimale, encore appelé **système à base DIX (10)** est composé de dix caractères nommés chiffres de 0 à 9.
Pour représenter une quantité supérieure à 9, il faut faire appel à un deuxième caractère. On obtient ainsi après 9 le nombre 10 qui représente une unité de rang supérieur, appelée «dizaine».
Les nombres de deux chiffres permettent de compter jusqu'à 99. Pour aller au-delà, il faut utiliser un troisième caractère.
Et ainsi de suite puisque la suite des nombres est illimitée. Cependant, dans beaucoup d'applications, le format est limité à un certain nombre de caractères. Ainsi un compteur kilométrique d'une voiture est limité à 5 chiffres. Il peut donc représenter une distance parcourue de 00 000 à 99 999 km.
Si nous dépassons le maximum, il repasse par 00 000. La capacité du compteur est : $10^5 - 1$. En effet :

$$10^5 = 100\,000 \text{ donc } 100\,000 - 1 = 99\,999$$
$$\text{où } 10^5 - 1 = 99\,999$$

Exemple :
Pour représenter 2076, nous plaçons les uns à côté des autres et en commençant par la gauche, le chiffre 2 puis le chiffre 0, le 7 et pour finir le 6. Cette représentation signifie en réalité que le nombre 2076 se compose de la manière suivante.
2 unités de mille ou 2×1000 ou 2×10^3
0 unité de cent ou 0×100 ou 0×10^2
7 unités de dix ou 7×10 ou 7×10^1
6 unités ou 6×1 ou 6×10^0.
(Rappel $A^0 = 1$)
et le nombre N peut s'écrire :

$$N = 2076 = 2 \times (10)^3 + 0 \times (10)^2 + 7 \times (10)^1 + 6 \times (10)^0$$

et en désignant par «b» la base (qui vaut 10 dans le système décimal)

$$N = 2076 = 2 \times (b)^3 + 0 \times (b)^2 + 7 \times (b)^1 + 6 \times (b)^0$$

D'une manière plus générale, un nombre N s'exprime dans le système décimal sous la forme :

$$N = a_n(10)^n + a_{n-1}(10)^{n-1} + \dots\dots + a_i(10)^i + \dots\dots + a_1(10)^1 + a_0(10)^0$$

ou

$$N = a_n(b)^n + a_{n-1}(b)^{n-1} + \dots\dots + a_i(b)^i + \dots\dots + a_1(b)^1 + a_0(b)^0$$

que l'on écrit :

$$n = a_n\, a_{n-1} \dots\dots a_i \dots\dots a_1\, a_0$$

expression dans laquelle chaque chiffre a_i ne peut que prendre une valeur entière comprise entre 0 et 9.
La position d'un chiffre «a_i» représente un «poids» égal à $(10)^i$ ou $(b)^i$.
Toutes ces notions de numération sont très importantes car elles restent **valables quelle que soit la base utilisée.**
Dans les systèmes logiques, on utilise la base b = 2, dans les calculateurs, ce sera la base b = 8 tandis que dans les systèmes microprocesseurs on utilise la base 16.

III.2. Le système de numération BINAIRE

La **base binaire** ne comporte que 2 caractères, représentés usuellement par «0» et «1» qu'il ne faut pas confondre avec les chiffres 0 et 1 du système décimal.
Dans la base binaire, le premier nombre est le nombre nul «0», le second est le nombre «1».
A partir du suivant, le problème se pose : comment peut-on représenter dans le système binaire l'équivalent du nombre 2 décimal ?
Nous savons, dans le système décimal, que pour passer de 9 à 10, nous avons utilisé un deuxième caractère. Nous ferons de même en binaire. Ainsi, pour représenter le nombre supérieur à 1 (binaire), nous ajoutons un deuxième caractère, qui représente une unité de rang supérieur.
Donc, nous dirons que 2 (décimal) s'écrit 10 en binaire.
Comme pour les nombres décimaux, nous inscrivons d'abord le chiffre des unités puis **à la gauche, l'unité de rang supérieur.**
Le chiffre 3 décimal s'écrit 11 en binaire puisqu'il suffit d'ajouter une unité à 10 (binaire) qui vaut 2 en décimal.
La quatrième unité est 8 et représente 2^3 ou $(b)^3$.
....
La nième unité représente 2^n ou $(b)^n$.
On appelle «POIDS» le nombre représenté par une unité quel que soit son rang.
Par exemple, le nombre 5 s'écrit 101 (binaire). Le premier caractère (le plus à gauche) représente les unités de «poids» le plus fort.
Dans le cas présent, la «présence» de ce poids (symbolisé par un 1) représente la quantité 4 en décimal.
Le second caractère «0» signifie que la quantité de poids est inférieure, dans ce cas, 2 n'est pas inclus dans le nombre. Par contre, le poids le plus faible (soit 1) est inclus.
En faisant la «somme» de tous ces «poids» (et on trouve une certaine analogie avec la balance de nos grands-mères), nous obtenons :

$$1\;0\;1 = 1 \times (2)^2 + 0(2)^1 + 1(2)^0 = 1 \times 4 + 0 \times 2 + 1 \times 1 = 4 + 0 + 1 = 5$$

Nous retrouvons ainsi une décomposition analogue à celle que nous avons étudiée pour les nombres décimaux, c'est-à-dire :

$$N = a_n(b)^n + a_{n-1}(b)^{n-1} + \dots\dots + a_i(b)^i + \dots\dots a_1(b)^1 + a_0(b)^0$$

Le chiffre 4 décimal ne peut pas s'exprimer avec les deux caractères, il faut ajouter une unité de rang 3 et 4 (décimal) = 100 (binaire).
Combien de nombres décimaux peut-on représenter avec 3 caractères binaires ?
Etablissons le tableau (figure 144) qui donne l'équivalence d'un nombre binaire avec le nombre décimal qu'il représente.

BINAIRE	DECIMAL
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Fig. 144

La figure 144 nous montre qu'avec 3 bits on peut représenter les 8 premiers nombres décimaux (0 à 7) au-delà, il faut rajouter une quatrième unité, qui s'écrira 1000 en binaire et vaudra 8 en décimal.

En résumé, dans le système binaire :

— la première unité est 1 et représente 2^0 ou $(b)^0$

— la deuxième unité est 2 et représente 2^1 ou $(b)^1$

— la troisième unité est 4 et représente 2^2 ou $(b)^2$.

Le tableau de la figure 145 donne pour les dix premières puissances de «2» la représentation du nombre binaire et son équivalence décimale.

2^n	Nd	N en binaire
2^0	1	1
2^1	2	10
2^2	4	100
2^3	8	1000
2^4	16	10000
2^5	32	100000
2^6	64	1000000
2^7	128	10000000
2^8	256	100000000
2^9	512	1000000000

Fig. 145

IV. OPERATIONS LOGIQUES ET ARITHMETIQUES

IV.1. Introduction

Les opérations logiques de base exécutables par le microprocesseur Z80 sont : ET, OU et OU exclusif.

Les opérations arithmétiques de base exécutables par le microprocesseur Z80 sont :

— Addition avec ou sans report

— Soustraction avec ou sans report

— Comparaison

— Incrémentation et décrémentation.

Nous ne reviendrons pas sur l'aspect purement logique des opérateurs puisque les principes ainsi que les tables de vérité ont été rappelées dans le paragraphe 2.

Les opérations arithmétiques ne présentent aucune difficulté ; nous dirons simplement qu'«incrémenter» consiste à ajouter une unité et que «décrémenter» consiste à en retrancher une.

Une opération logique (ET, OU, OU exclusif) s'effectue toujours bit à bit entre un opérande désigné par «s» et le contenu de l'accumulateur A. Le résultat de cette opération est replacé dans l'accumulateur.

Une opération arithmétique (addition ou soustraction) s'effectue toujours entre la quantité contenue dans l'accumulateur et un opérande désigné par «S». Le résultat de cette opération est replacé dans l'accumulateur.

Les opérations d'incrémentation ou de décrémentation s'effectuent toujours directement sur la donnée d'un registre désigné ou celle d'une case mémoire pointée par l'une des paires de registres HL, IX + d ou IY + d.

Au résultat de l'opération logique ou arithmétique s'ajoute un second effet dont il faudra savoir tenir compte : c'est le **positionnement des indicateurs en fonction du résultat obtenu (registre F).**

Pour les **trois opérations logiques,** les règles sont les suivantes :

— Indicateurs «C» (carry ou report) et «N» (indique que l'opération précédente était une soustraction) sont remis à «0».

— Indicateur «Z» (zéro) est mis à «1» quand le résultat de l'opération entraîne une mise à «0» de tous les bits de l'accumulateur.

— Indicateur «P/V» est mis à 1 quand le nombre de 1 de l'accumulateur est pair. Il est à 0 quand le nombre de 1 est impair (indicateur P). Fonction PARITE.

— Indicateur «S» (indicateur de signe) est la recopie du bit 7. Dans les opérations sur les nombres relatifs, la mise en 1 de ce bit signifie que le nombre est négatif.

— Indicateur «H» est à «1» quand une retenue provient des 4 bits les moins significatifs.

Pour les **opérations arithmétiques,** les règles sont identiques excepté pour les indicateurs suivants :

— Indicateur «C» est mis à 1 quand un report s'échappe du bit 7. Sinon remis à 0.

— Indicateur «P/V» est mis à 1 quand un dépassement de capacité survient (indicateur V). Fonction DEBORDEMENT.

— Indicateur «N» est mis à 1 dans le cas des opérations de soustraction.

Dans l'étude des codes opérations, nous ne reviendrons pas sur ces règles concernant les indicateurs.

IV. 2. ET logique

Le code opératoire général de l'opération logique ET est :

$$A \leftarrow A \wedge s$$

($\wedge$ signifie ET logique)

(Nota : le signe «$\wedge$» est équivalent au symbole «X»)

dans lequel «s» désigne l'un des opérandes suivants :

r : désigne l'un des registres A, B, C, D, E, H ou L

n : une donnée de 8 bits

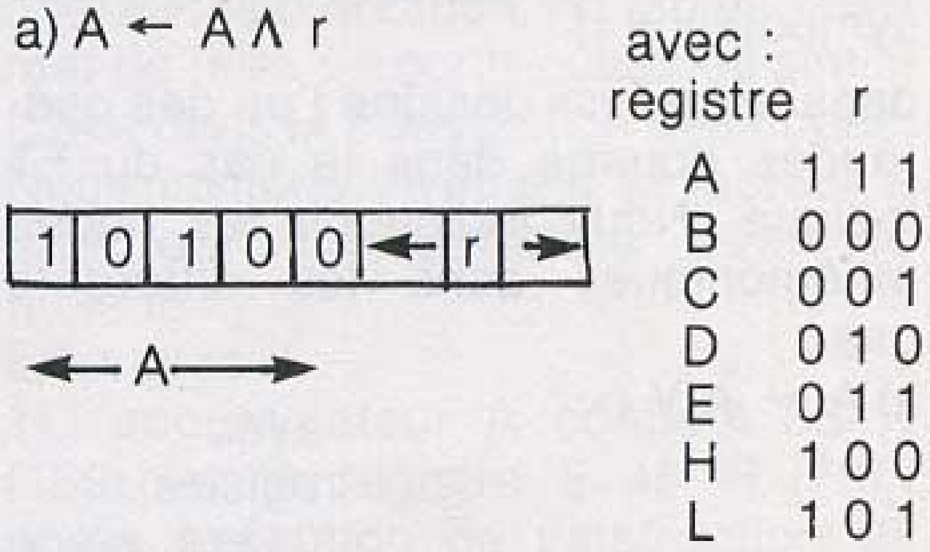
(HL) : le contenu de l'emplacement pointé par HL

(IX + d) : le contenu de l'emplacement pointé par IX auquel s'ajoute le déplacement «d»

(IY + d) : le contenu de l'emplacement pointé par IY auquel s'ajoute le déplacement «d».

Suivant la provenance du byte que représente «s», les codes mnémoniques sont les suivants :

a) $A \leftarrow A \wedge r$



Exemple :

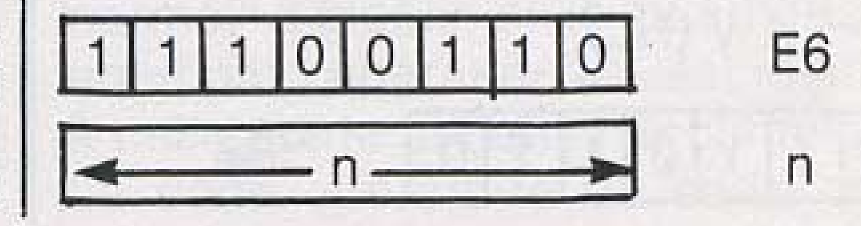
Effectuer un ET logique entre le contenu de l'accumulateur et le registre D, a pour code A2.

Si A contenait 1001 1101 et le registre D1010 0111, après exécution de l'instruction A2, l'accumulateur contient 1000 0101.

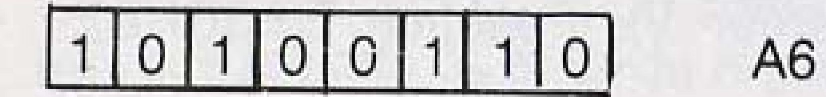
Le bit «S» est mis à 1. Les autres remis à «0».

b) $A \leftarrow A \wedge n$

La donnée «n» suit immédiatement le code opératoire.



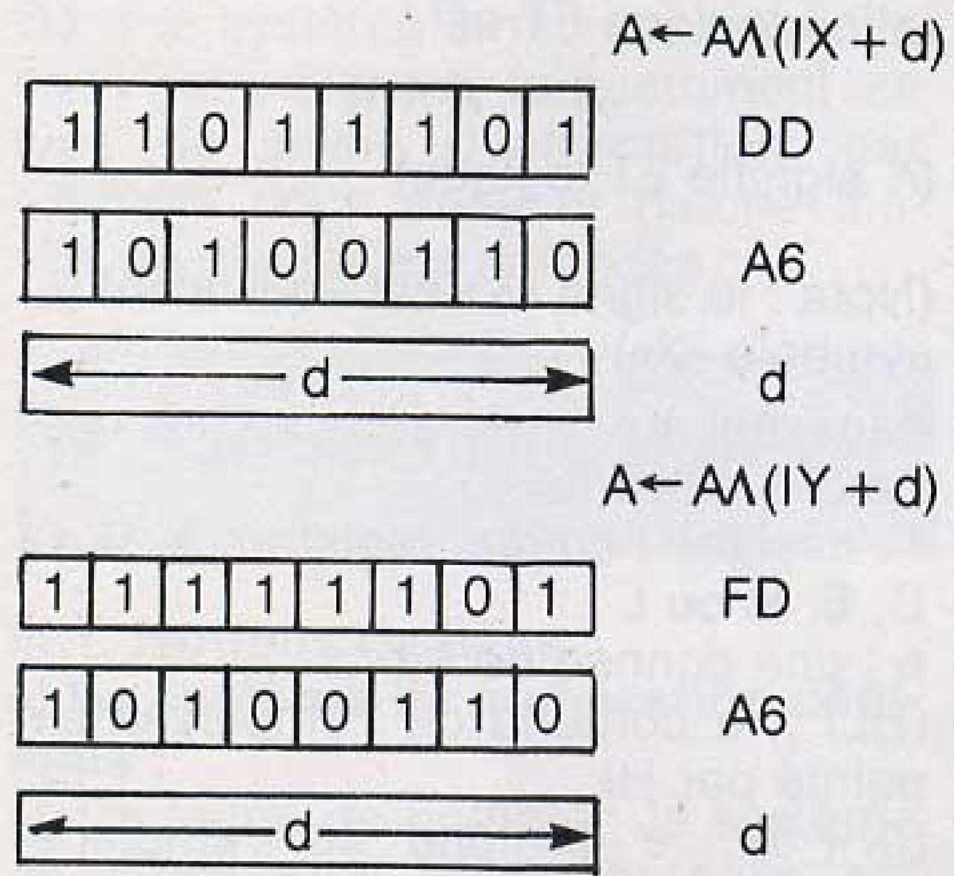
c) A ← A ∧ (HL)
L'opérande «s» est la donnée contenue dans la case mémoire pointée par la paire de registres HL.



A6

d) A← A∧(IX + d)ou A← A∧(IY + d)
L'opérande «s» est la donnée contenue dans le case mémoire dont l'adresse est le contenu de la paire de registres IX et IY auquel est ajouté le déplacement «d» spécifié dans le code opératoire.

A← A∧(IX + d)



IV.3. OU logique

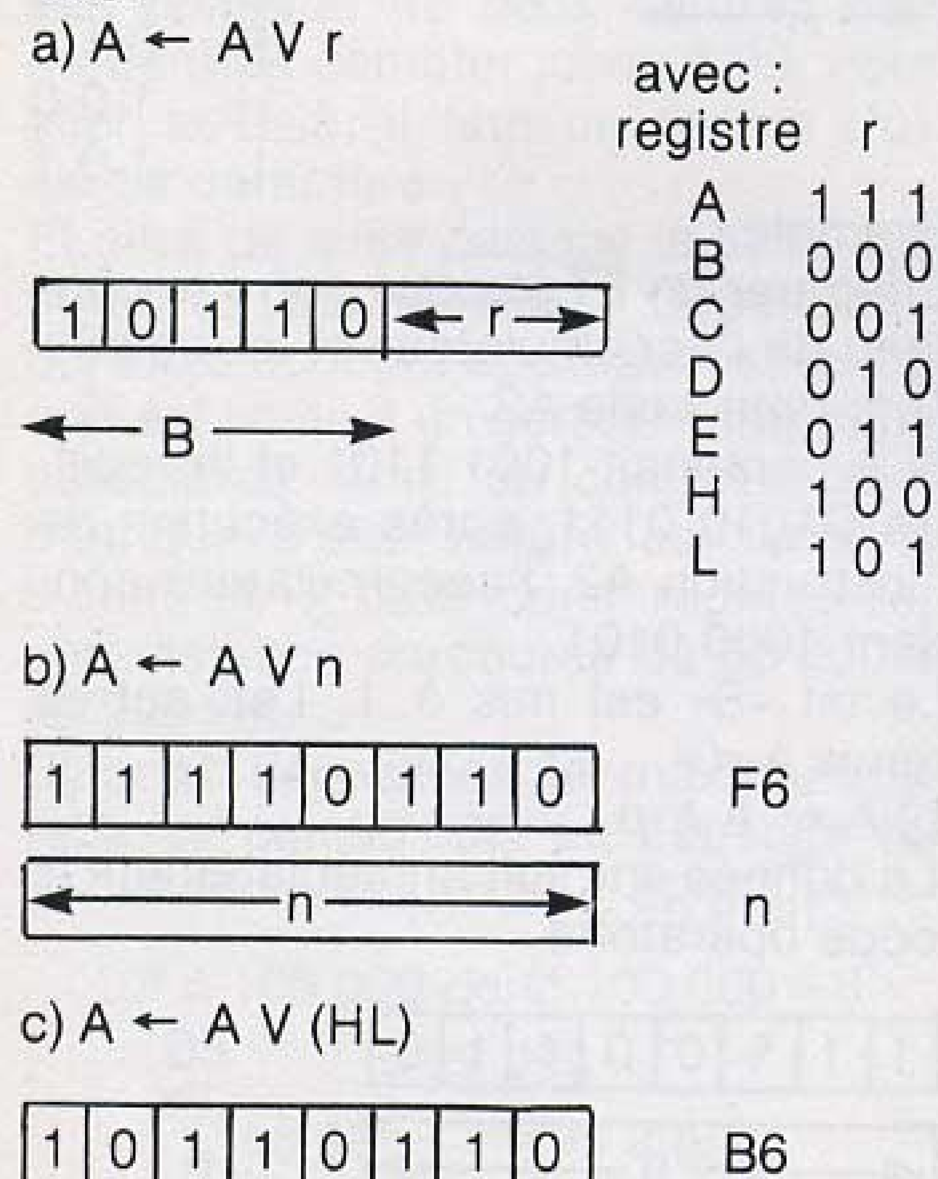
Le code opératoire général de l'opération logique OU est :

A ← A V s

(V signifie OU logique)

(Nota : le signe «V» est équivalent au symbole « + »).

dans lequel «s» désigne l'un des opérandes comme dans le cas du ET logique. Nous indiquons les codes mnémoniques dans les différents cas :



a) A ← A V r

avec : registre	r
A	1 1 1
B	0 0 0
C	0 0 1
D	0 1 0
E	0 1 1
H	1 0 0
L	1 0 1

1 0 1 1 0 ← r →

← B →

b) A ← A V n

1 1 1 1 0 1 1 0 — F6

← n → — n

c) A ← A V (HL)

1 0 1 1 0 1 1 0 — B6

d) A← AV(IX + d)ou A← AV(IY + d)

A ← A V (IX + d)

1 1 0 1 1 1 0 1 — DD

1 0 1 1 0 1 1 0 — B6

← d → — d

A ← A V (IY + d)

1 1 1 1 1 1 0 1 — FD

1 0 1 1 0 1 1 0 — B6

← d → — d

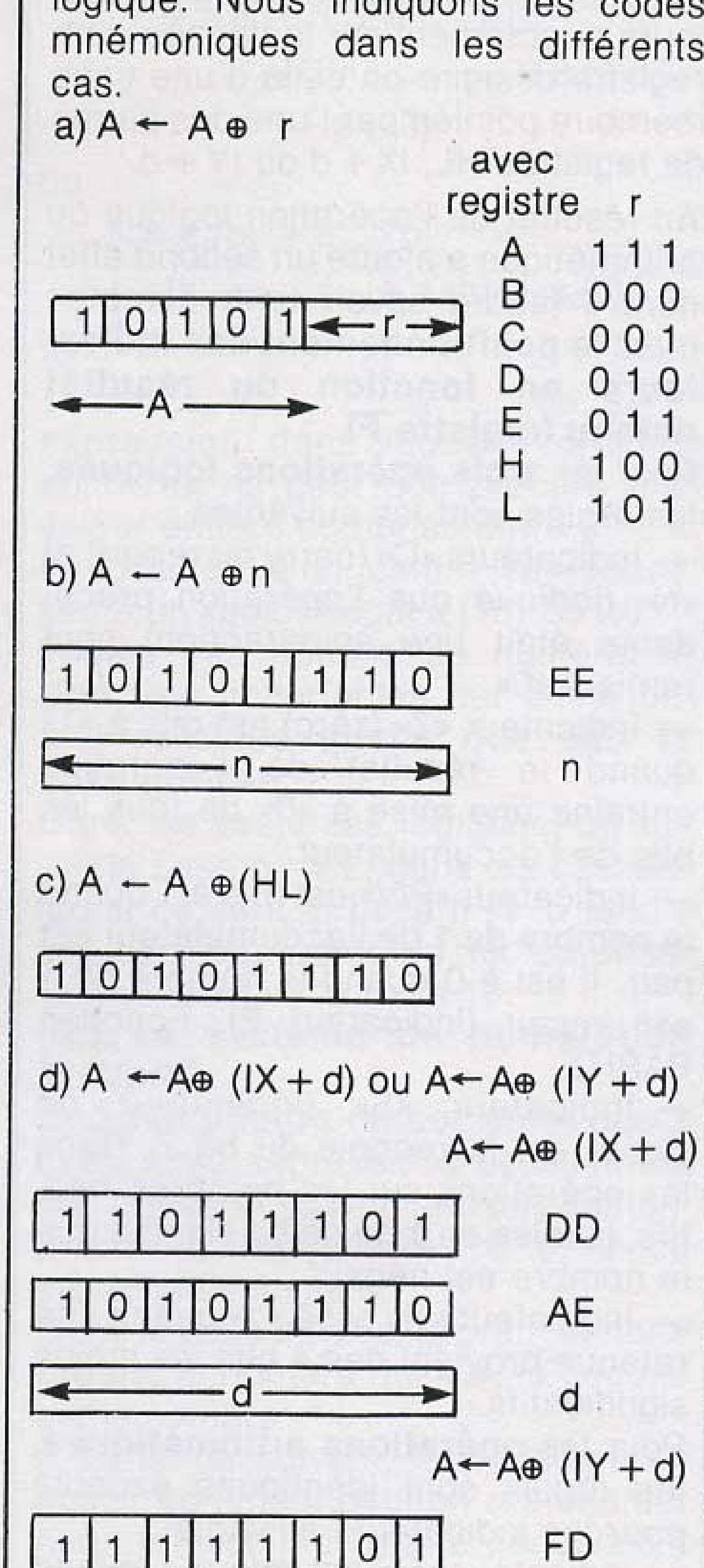
IV.4. OU exclusif

Le code opératoire général de l'opération logique OU exclusif est :

A ← A ⊕ s

(⊕ signifie OU exclusif)

dans lequel «s» désigne l'un des opérandes comme dans le cas du ET logique. Nous indiquons les codes mnémoniques dans les différents cas.



a) A ← A ⊕ r

avec registre	r
A	1 1 1
B	0 0 0
C	0 0 1
D	0 1 0
E	0 1 1
H	1 0 0
L	1 0 1

1 0 1 0 1 ← r →

← A →

b) A ← A ⊕ n

1 0 1 0 1 1 1 0 — EE

← n → — n

c) A ← A ⊕(HL)

1 0 1 0 1 1 1 0

d) A ← A⊕ (IX + d) ou A← A⊕ (IY + d)

A← A⊕ (IX + d)

1 1 0 1 1 1 0 1 — DD

1 0 1 0 1 1 1 0 — AE

← d → — d

A← A⊕ (IY + d)

1 1 1 1 1 1 0 1 — FD

1 0 1 0 1 1 1 0 — AE

← d → — d

IV.5. Addition arithmétique (sans report)

Le code opératoire général de l'opération arithmétique addition est :

A← A + s ou ADD A, s

dans lequel «s» désigne l'un des opérandes suivants :

r : désigne l'un des registres A, B, C, D, E, H ou L

n : une donnée de 8 bits

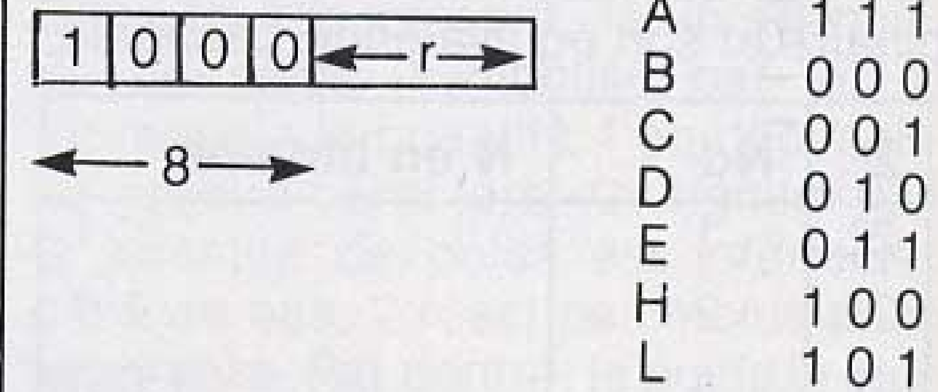
(HL) : le contenu de l'emplacement pointé par (HL)

(IX + d) : le contenu de l'emplacement pointé par IX auquel s'ajoute le déplacement «d»

(IY + d) : le contenu de l'emplacement pointé par IY auquel s'ajoute le déplacement «d».

Suivant la provenance du byte que représente «s», les codes mnémoniques sont les suivants :

a) A← A + r ou ADD A, r



Exemple :

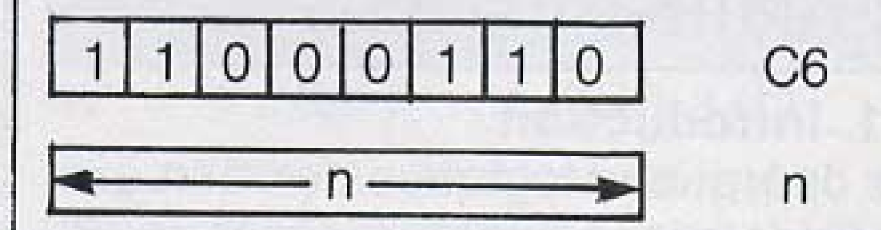
L'instruction «additionner au contenu de l'accumulateur A le contenu du registre H» a pour code 84.

Si A contient 3 BH $(59)_d$ et le registre H contient 79 H $(121)_d$ après exécution de l'instruction 84, l'accumulateur contient B4 H $(180)_d$.

L'indicateur «S» est à 1 tandis que tous les autres sont à «0».

b) A← A + n ou ADD A, n

La donnée «n» à additionner suit immédiatement le code opération.



Exemple :

Le registre A contient 43 H $(67)_d$, après exécution de C6 1E, le registre A contient 61 H $(97)_d$. Aucun indicateur n'est à 1.

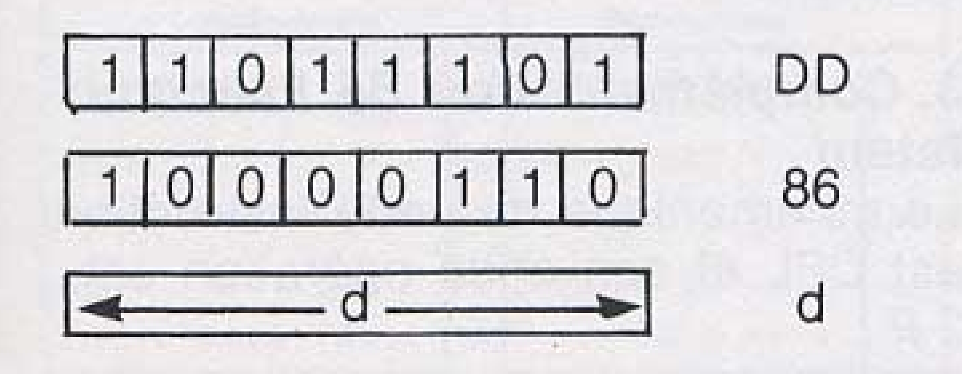
c) A← A + (HL) ou ADD A, (HL)

La donnée à additionner est le contenu de l'emplacement pointé par HL. Le code opération est :

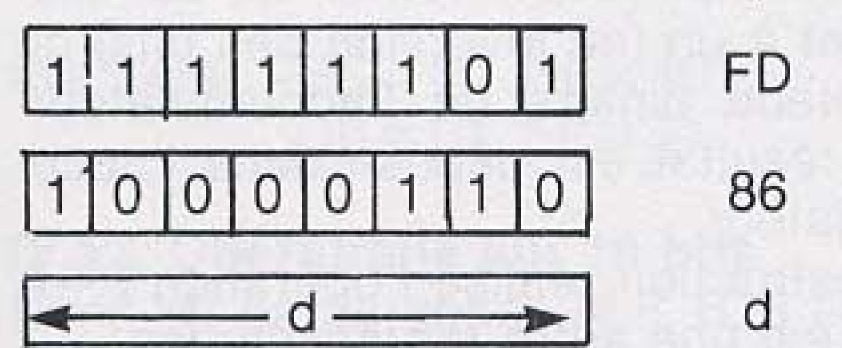
1	0	0	0	0	1	1	0	86

Exemple :
Le registre A contient DDH $(221)_d$, le contenu de la paire de registres HL est 18FE H et le contenu de l'emplacement 18FE H est 4B H.
Après exécution de l'instruction 86, le contenu de A est de 2 BH $(43)_d$.
Les indicateurs P/V et C sont mis à 1.
d) A←A+(IX+d) ou A←A+(IY+d)
La donnée à additionner est le contenu de l'emplacement pointé par le contenu de la paire de registres IX ou IY auquel est ajouté le déplacement «d».
Les codes opératoires sont :

A←A+(IX+d)



A←A+(IY+d)



Exemple :
L'accumulateur contient 32 H $(50)_d$, le registre d'index IY contient 18 00 et l'emplacement mémoire 18 40 H contient 19 H ; après exécution de FD 86 40 ou A←A+(IY+40) l'accumulateur contient 4B H. Tous les indicateurs sont à «0».

IV.6. Addition arithmétique avec report

Le code opératoire général de l'opération arithmétique addition avec report (c'est-à-dire en tenant compte du bit C du registre F) est :

A←A+s+Cy

L'opération ADC (addition avec report) est identique à l'opération ADD, si ce n'est que dans la première il est tenu compte du bit C qui est 0 ou 1.
Nous présentons sous la forme d'un tableau comparatif les codes mnémoniques des différentes instructions.
Remarque :
Cette représentation a l'avantage de faire ressortir les différentes similitudes dans l'élaboration des codes mnémoniques. Essayez de les trouver vous-même.

ADD A, s A←A+s	ADC A, s A←A+s+Cy
ADD A, r 10 000 ←r→	ADC A, r 10 001 ←r→
ADD A, n 11 000 110 ←n→	ADC A, n 11 001 110 ←n→
ADD A, (HL) 10 000 110	ADC A, (HL) 10 001 110
ADD A, (IX+d) 11 111 101 10 000 110 ←d→	ADC A, (IX+d) 11 111 101 10 001 110 ←d→
ADD A, (IY+d) 11 111 101 10 000 110 ←d→	ADC A, (IY+d) 11 111 101 10 001 110 ←d→

Tableau I

IV.7. Positionnement de l'indicateur C

Nous venons de voir dans les additions avec report que l'indicateur C (registre F) pouvait entrer dans l'exécution des opérations arithmétiques entre autres. Il faut donc pouvoir contrôler son contenu. Pour cela le Z80 possède deux instructions :
a) mise à «1» de l'indicateur «C»
b) complémentation de l'indicateur «C».

a) Mise à 1 de l'indicateur C «SCF»

Le mnémonique de cette instruction est «SCF» ou C←1 et son code opération : 37.
Description :
L'indicateur de report C (bit 0 du registre F) est mis à 1 après exécution de cette instruction.

b) Complémentation de l'indicateur «C»

Le mnémonique de cette instruction est «CCF» ou C←C et son code opération : 3F.
Description :
Le bit 0 du registre F (indicateur C) est remplacé par son complément. S'il était «1» avant l'exécution de l'instruction, il devient «0». S'il était «0», il devient «1».

IV.8. Soustraction arithmétique (sans report)

Le code opératoire général de l'opération arithmétique soustraction est :

A←A−s ou SUB A, s

dans lequel «s» désigne l'un des opérandes comme indiqué dans l'addition arithmétique.
Les règles de positionnement des indicateurs sont identiques à l'exception de l'indicateur N qui est systématiquement mis à 1 puisqu'il s'agit d'une opération de soustraction.
Les codes mnémoniques sont donnés par le tableau II.

IV.9. Soustraction arithmétique (avec report)

Le code opératoire général de l'opération arithmétique soustraction avec report est :

A←A−s−Cy ou SBC A, s

dans lequel «s» désigne l'un des opérandes.
Les règles de positionnement des indicateurs sont identiques à l'opération soustraction arithmétique.
Les codes mnémoniques sont donnés par le tableau I.

SUB A, s A←A−«s»		SBC A, s A←A−«s»−Cy	
SUB r 1001 0←r→	9 –	SBC r 1001 1←r→	9 –
SUB n 1101 0110 ←n→	D6 n	SBC n 1101 1110 ←n→	DE
SUB (HL) 1001 0110	96	SBC (HL) 1001 1110	9E
SUB (IX+d) 1101 1101 1001 0110 ←d→	DD 96	SBC (IX+d) 1101 1101 1001 1110 ←d→	DD 9E
SUB (IY+d) 1111 1101 1001 0110 ←d→	FD 96	SBC (IY+d) 1111 1101 1001 1110 ←d→	FD 9E

Tableau II

Remarque :
Comme dans la présentation des opérations d'addition, ce tableau permet de faire ressortir quelques similitudes dans la constitution des codes. Nous invitons vivement le lecteur à effectuer un parallèle avec le tableau I.
Exemples :
1) L'accumulateur A contient 85 H $(133)_d$ et le registre E 48 H $(75)_d$, après exécution de l'instruction 93 (A←A−E), le contenu de A sera 3A H $(58)_d$. Les indicateurs N et H sont à «1», les autres à «0».
2) L'accumulateur A contient 3A $(186)_d$, l'indicateur C est à 1, la paire de registres HL contient 20 FE H et la case mémoire 20 FE, 55 H, après exécution de l'instruction 9E, le contenu de l'accumulateur sera 64 H $(100)_g$. Seul l'indicateur N sera positionné.

IV.10. Comparaison

Cette instruction de «comparaison» permet d'établir si la quantité désignée par l'opérande «s» est identique à celle contenu dans l'accumulateur. Le code opératoire de cette opéra-

tion est : A – s dans lequel «s» désigne l'un des opérandes comme indiqué dans l'addition arithmétique.
Cette opération est similaire à la soustraction, à ceci près que le résultat n'apparaît pas dans l'accumulateur, mais dans le registre F. Si la comparaison est vraie, le bit Z est 1 (puisque A – s donne 0) tandis que si la comparaison est différente Z = 0 ; c'est cet indicateur qui donne le résultat de la comparaison.
Les autres indicateurs sont positionnés comme dans la soustraction.
Les codes mnémoniques sont donnés par le tableau suivant :

CP A – s		
CP r	1 0 1 1 1 ←r→	B
CP n	1111 1110	FE
	←n→	
CP (HL)	1011 1110	BE
CP (IX + d)	1101 1101	DD
	1011 1110	BE
	←d→	
CP (IY + d)	1111 1101	FD
	1011 1110	BE
	←d→	

Tableau III

Exemple :
L'accumulateur A contient FA H et le registre B contient 86 H, après exécution de l'instruction B8 (CP A et B), tous les indicateurs sont à 0 sauf N.

IV.11. Opérations particulières sur l'accumulateur

Nous allons décrire trois instructions particulières qui ne portent que sur le contenu de l'accumulateur et qui facilitent les opérations arithmétiques.

1. Ajustement décimal de l'accumulateur «DAA»

Le mnémonique de cette instruction est DAA et son code opération : 27.
Description :
Pour bien comprendre cette instruction, nous allons traiter un exemple. Jusqu'à présent nous avons dit que les opérations arithmétiques ne pouvaient s'effectuer que sur des quantités exprimées en binaire.

L'instruction DAA permet de réaliser des opérations arithmétiques sur des quantités exprimées en B.C.D (Données et résultat).

Soit à additionner les nombres BCD 59 et 22, l'arithmétique décimale donne :

```
   59
 + 22
 ----
   81
```

L'arithmétique binaire donne :

```
   0101  1001
 + 0010  0010
 ------------
   0111  1011 = 7B
```

Le résultat 7B est équivoque : il fait apparaître notamment un caractère (B) qui est «interdit» en BCD.
Ajoutons 06 (0000 0110) au résultat.
Ce qui donne :

```
   0111  1011
 + 0000  0110
 ------------
   1000  0001 = 81
```

Le résultat (code BCD) 81 est maintenant correct.

Conclusion :
L'instruction DAA (Decimal Adjust Accumulator) a pour but d'effectuer conditionnellement les corrections nécessaires sur le contenu de l'accumulateur après les opérations arithmétiques (addition et soustraction) pour obtenir un résultat en BCD.

La valeur corrective ainsi ajoutée au cours de l'instruction DAA, dépend des 2 demi-octets (ou quartets) contenus dans A, comme l'indique le tableau IV.
Appplication :

```
1800  Load A, 59 H   3E  59
1802  ADD A, 22 H    C6  22
1804  DAA            27
1805  HALT           76
```

Après l'exécution de ce programme, le contenu de A est 81.

2. Valeur de signe opposé dans l'accumulateur

Le mnémonique de cette instruction est NEG et son code opération est : ED 44.

Description :
Cette instruction forme le complément à deux (ou la valeur opposée) de la quantité contenue dans l'accumulateur. Le résultat est déposé dans l'accumulateur.
L'instruction réalise l'opération A←0 – A qui est une autre représentation.

Exemple :

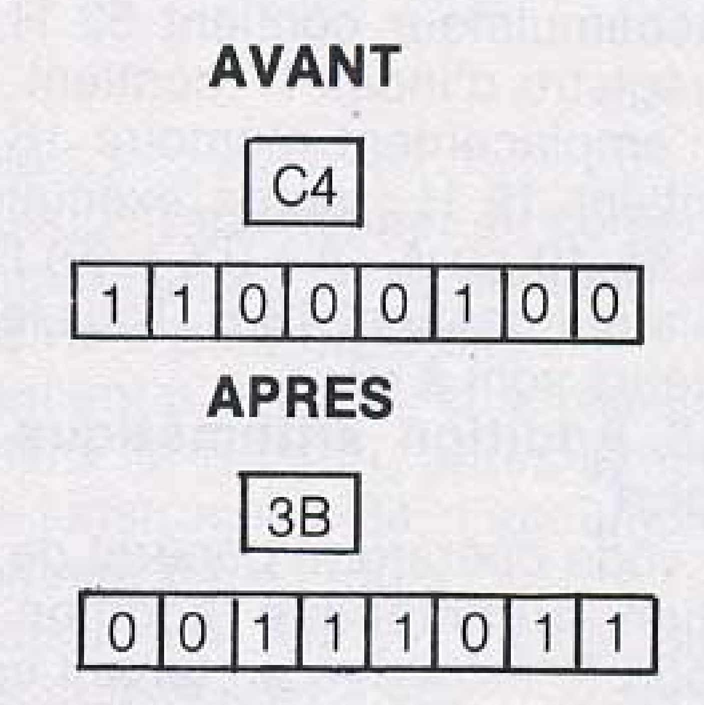
AVANT	APRES
A : C4	A : 3C

Opération	Indic. C Avant DAA	Valeur du quartet de poids fort	Indice H H Avant DAA	Valeur du quartet de poids faible	Quantité corrective ajoutée à «A»	Indice C Après DAA
	0	0-9	0	0-9	00	0
	0	0-8	0	A-F	06	0
ADD	0	0-9	1	0-3	06	0
ADC	0	A-F	0	0-9	60	1
INC	0	9-F	0	A-F	66	1
	0	A-F	1	0-3	66	1
	1	0-2	0	0-9	60	1
	1	0-2	0	A-F	66	1
	1	0-3	1	0-3	66	1
SUB	0	0-9	0	0-9	00	0
SBC	0	0-8	1	6-F	FA	0
DEC	1	7-F	0	0-9	A0	1
NEG	1	6-F	1	6-F	9A	1

Tableau IV

3. Complémentation de l'accumulateur

Le mnémonique de cette instruction est CPL et son code opération est : 2 F.
Description :
Cette instruction forme le complément à un (ou inversion des bits) du contenu binaire de l'accumulateur. Le résultat est déposé dans l'accumulateur.
L'instruction réalise l'opération A←A qui est une autre représentation.
Exemple :



IV.12. Incrémentation et décrémentation

Les opérations «INC» et «DEC» ne sont en fait que des cas particuliers des opérations arithmétiques ADD et SUB dans lesquelles la quantité à additionner (INC) ou à retrancher (DEC) est la valeur unitaire : 1.
Le code général de l'instruction est :

INC m ou DEC m

dans lequel m désigne l'un des opérandes suivant : soit l'un des registres soit l'un des emplacements mémoires.
Les codes mnémoniques sont donnés par le tableau suivant :

a) «m» représente une quantité de 1 octet

INC m m ← m + 1		DEC m m ← m − 1	
INC r 00←r→100		DEC r 00←r→101	
INC (HL) 0011 0100	34	DEC (HL) 0011 0101	35
INC (IX + d) 1101 1101 0011 0100 ←d→	DD 34	DEC (IX + d) 1101 1101 0011 0101 ←d→	DD 35
INC (IY + d) 1111 1101 0011 0100 ←d→	FD 34	DEC (IY + d) 1111 1101 0011 0101 ←d→	FD 35

b) Registres 16 bits :

Incrémentation		Décrémentation	
INC «ss» 00ss 0011		DEC «ss» 00ss 1011	
INC IX 1101 1101	DD	DEC IX 1101 1101	DD
0010 0011	23	0010 1011	2B
INC IY 1111 1101	FD	DEC IY 1111 1101	FD
0010 0011	23	0010 1011	2B

Tableau V

avec «ss» : BC = 00 HL = 10
DE = 01 SP = 11

IV.13. Opérations sur 16 bits

Dans les paragraphes IV.5, 6, 8 et 9, nous avons étudié les opérations arithmétiques sur 8 bits. Le résultat était ensuite placé dans l'accumulateur.

Il est possible de réaliser des opérations arithmétiques sur 16 bits entre paires de registres. Danc ce cas, le résultat n'est plus déposé dans l'accumulateur (8 bits) mais dans la paire de registres HL ou l'un des registres index IX ou IY.

1. Addition sur 16 bits sans report

0 0 r r 1 0 0 1

avec rr : BC = 00 IX = 10
DE = 01 SP = 11

Indicateurs :
C est positionné par le report du bit 15
H est positionné en fonction du bit 7.

IY←IY + rr ou ADD IY, rr

Description :
Le contenu de la paire de registres «rr» (BC, DE, IY ou SP) est ajouté au contenu du registre index IY et le résultat déposé dans IY.
Le code opératoire est :

1 1 1 1 1 1 0 1   FD

0 0 r r 1 0 0 1

avec rr : BC = 00 IY = 10
DE = 10 SP = 11

Indicateurs :
C est positionné par le report du bit 15.
H est positionné en fonction du bit 7.

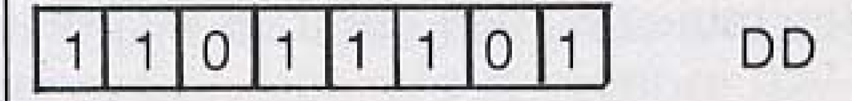
a) HL←HL + ss ou ADD HL, ss
Description :
Le contenu de la paire de registres «ss» (BC, DE, HL Ou SP) est ajouté au contenu de la paire de registres HL et le résultat est déposé dans HL.
Le code opératoire est :

0 0 s s 1 0 0 1

avec ss : BC = 00 HL = 10
DE = 01 SP = 11

Indicateurs :
L'indicateur C est positionné par le report du bit 15 sinon remis à 0.
L'indicateur H est positionné en fonction du bit 7.

b) IX←IX + rr ou ADD IX, rr
Description :
Le contenu de la paire de registres «rr» (BC, DE, IX ou SP) est ajouté au contenu du registre index IX et le résultat déposé dans IX.
Le code opératoire est :


1 1 0 1 1 1 0 1   DD

2. Addition sur 16 bits avec reports. ADC HL←HL + rr + C

Description :
Le contenu de la paire de registres «ss» (BC, DE, HL ou SP) est ajouté au contenu de la paire de registres puis la valeur de l'indicateur C. Le résultat est ensuite déposé dans HL.
Le code opératoire de HL←HL + ss + C est :

1 1 1 0 1 1 0 1   ED

0 1 s s 1 0 1 0

avec ss : BC = 00 HL = 10
DE = 01 SP = 11

Les indicateurs S, Z, P/V et C sont positionnés en fonction du résultat. N est à zéro. H est positionné en fonction du bit 11.

3. Soustraction sur 16 bits avec report

Description :
Le contenu de la paire de registres «ss» (BC, DE, HL ou SP) auquel est ajoutée la valeur de l'indicateur C est soustrait du contenu de la paire de registres HL.
Le résultat est ensuite déposé dans HL.
Le code opératoire de HL←HL − ss − C.

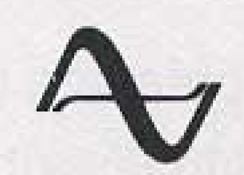
1 1 1 0 1 1 0 1   ED

0 1 s s 0 0 1 0

avec ss : BC = 00 HL = 10
DE = 01 SP = 11

Les indicateurs S, Z, P/V et C sont positionnés en fonction du résultat. N est à 1, H est positionné en fonction du bit 12.

Philippe Duquesne



COURS PRATIQUE DE MICROPROCESSEUR AVEC LE MICROPROFESSOR MPF-IB

HUITIEME PARTIE

Les modes d'adressage

SOMMAIRE

I. INTRODUCTION

Avant de poursuivre l'étude proprement dite de la programmation d'un microprocesseur, nous allons présenter les divers modes d'adressage. Pour définir une instruction, il ne suffit pas de **déterminer l'opération à réaliser,** il faut bien souvent **indiquer les adresses de la donnée d'origine ainsi que celle de la destination** du résultat.

D'autre part, nous étudierons dans la prochaine leçon, les instructions de «sauts de programme». Selon **les directives du programmeur** et si certaines conditions sont remplies, on assiste à des «déroutements» de programme. Le microprocesseur **effectue un «saut» à une adresse différente du déroulement classique.** L'instruction à exécuter n'est plus la suivante, mais **au contraire elle se situe à un tout autre emplacement.** Dans ce cas, le compteur de programme est «chargé» avec une nouvelle adresse.

Nous reviendrons sur cette technique ultérieurement.

II. MODE D'ADRESSAGE

Le microprocesseur Z-80 ne comporte pas moins de 10 modes d'adressage. En réalité, nous les avons presque tous rencontrés dans la leçon précédente, mais sans y faire allusion directement.

II.1. Adressage immédiat

L'adresse qui contient l'opérande est celle qui suit **«immédiatement»** le code opération.

Exemple :

Additionner au contenu de l'accumulateur la quantité «3A».

PC (adresse du programme)	Contenu
1800	C6
→ 1801	3A
1802	

L'adresse de l'instruction C6

(ADD A ← A + n)

est 1800, celle de l'opérande «n» est 1801 : elle **suit immédiatement celle de l'instruction.**

II.2. Adressage étendu immédiat

Dans ce cas, l'opérande n'est plus constituée du seul octet qui **suit l'instruction,** mais des **«deux octets» consécutifs.**

Exemple 1

Charger la paire de registres HL avec 04 FE.

PC	Contenu	
1800	21	
← 1801	FE	(adresse de l'octet de poids faible)
← 1802	04	(adresse de l'octet de poids fort)

A noter que l'adresse (1801 dans l'exemple) qui suit immédiatement celle de l'instruction est celle de l'octet de poids faible de l'opérande et la suivante (1802) celle de l'octet de poids fort.

Exemple 2

Charger le registre d'index IY avec 30 00.

PC	Contenu	
1800	FD	
1801	21	
← 1802	00	(adresse de l'octet de poids faible)
← 1803	30	(adresse de l'octet de poids fort)

Dans cet exemple, l'instruction

LD IY ← 30 00

est constituée de 2 bytes : FD et 21. Dans ce cas les adresses de l'opérande sont 1802 et 1803.

III.3. Adressage étendu

Dans ce mode d'adressage, les deux octets consécutifs qui suivent immédiatement le code opératoire (1 ou 2 bytes) constituent **une adresse de 16 bits.** (Le poids faible étant le premier octet, le poids fort est le suivant). Deux cas se présentent :

1) Quand il s'agit **d'une instruction de saut,** les 2 octets représentent **l'adresse à partir de laquelle le programme se poursuit.**

2) Quand il s'agit **d'une donnée,** ils représentent l'adresse de **l'emplacement mémoire qui contient l'opérande** (1 octet).

Il ne faut pas confondre l'adressage étendu avec l'adressage étendu immédiat.

Exemple 1

Charger le contenu de l'accumulateur dans la case d'adresse 2300 H.

PC	Contenu	
1800	32	
→ 1801	00	(octet de poids faible de l'adresse)
→ 1802	23	(octet de poids fort de l'adresse)

ce qui s'écrit :

LD (2300 H) ← A

Exemple 2

Effectuer un saut à 1900 H.

PC	Contenu	
1800	C3	
→ 1801	00	(octet de poids faible de l'adresse)
→ 1802	19	(octet de poids fort de l'adresse)

Dans ce cas l'instruction qui sera exécutée après 1802 ne sera pas celle contenue dans 1803 **mais celle contenue dans 1900 H** ce qui s'écrit :

JP 1900 H

On notera dans ce dernier cas **l'absence de parenthèses.**

II.4. Adressage par registre

Dans ce mode d'adressage, l'instruction contient non seulement le code opération mais aussi **le registre concerné** qui peut être aussi bien **la source de l'opérande que la destination du résultat de l'opération.**

Exemple 1

Effectuer un OU logique entre le contenu de l'accumulateur et le contenu du registre E.

Ce qui s'écrit :

B3 ou A ← A + E (+ : ET logique)

Exemple 2

Décrémenter le contenu du registre L.

Ce qui s'écrit :

2 D ou L ← L − 1

Exemple 3

Charger le registre C avec le contenu du registre E.

Ce qui s'écrit :

4 B ou E ← C

II.5. Adressage indirect par registre

Dans ce mode d'adressage, l'adresse de l'opérande est le **contenu d'une paire de registres spécifiée** (16 bits). Ce qui revient à dire que la paire de registres (BC, DE ou HL) se comporte comme un «pointeur» d'adresse de la mémoire.

Exemple :

Comparer le contenu de l'accumulateur A avec le contenu de la case mémoire pointée par (HL).

Ce qui s'écrit :

BE ou CP A, (HL)

On notera la **présence de parenthèses** pour **(HL).**

II.6. Adressage indexé

L'adresse de l'opérande est dans ce cas constituée, comme dans le cas précédent, **du contenu d'un registre** (ici IX ou IY) **au contenu duquel on ajoute un déplacement «d».** **L'instruction contient une valeur arithmétique «d»** qui **est ajoutée au contenu du registre index spécifié** durant l'exécution du programme, **pour constituer l'adresse de l'opérande.**

Exemple :

Le registre d'index IY contient 20 00 H, l'instruction

LD (IY + 10 H), FE

chargera l'emplacement mémoire 20 00 + 10 = 20 10 H avec la quantité FE.

PC	Contenu	
1800	FD	
1801	36	
1802	10	déplacement
1803	FE	

II.7. Adressage implicite

Dans ce mode d'adressage, **l'un des registres du CPU** bien souvent l'accumulateur **est automatiquement impliqué à être la destination** du résultat.

Exemple :

Retrancher 23 H du contenu de l'accumulateur s'écrit :

D 6 23 H ou A ← A − 23 H

II.8. Adressage relatif

Le mode d'adressage relatif est essentiellement utilisé dans les instructions de branchement (conditionnel ou inconditionnel).

Pour que l'adressage soit valide, une règle limitant la distance du déplacement de l'instruction de branchement à la destination doit être respectée.

Cette règle est que **l'adresse de branchement doit être comprise dans les limites suivantes :**

$$(PC + 2) - 128 \leqslant D \leqslant (PC + 2) + 127$$

On notera que **la valeur du déplacement «D» est un nombre signé, donc compris entre + 127 ou − 128.**

II.9. Adressage modifié en page zéro

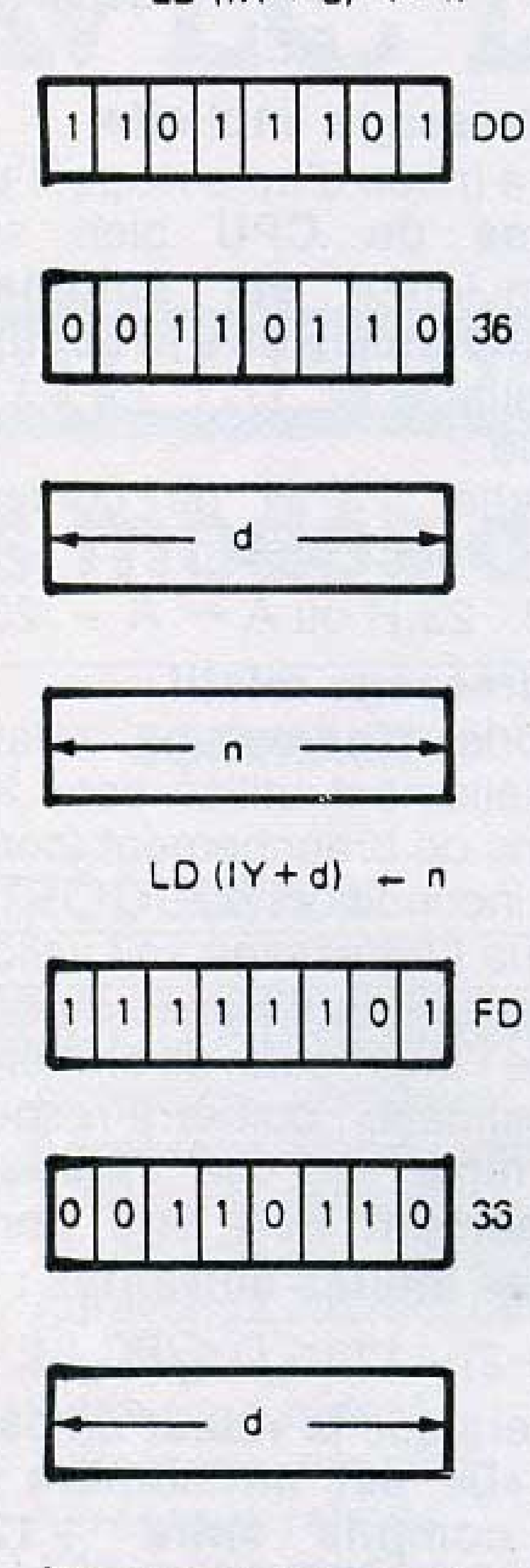
Comme dans le mode précédent, il s'agit d'un saut de programme. Il existe dans le Z-80 huit instructions d'un seul octet qui permettent de charger le compteur de programme avec l'un des huit octets qui constitue **le byte de poids faible d'une adresse en page zéro.** C'est-à-dire que le premier octet (poids fort) est 00 tandis que le second est spécifié dans l'instruction. Ainsi quand le programme exécute une instruction RESTART (comme on les appelle), il peut effectuer en un temps très court un saut à la première adresse d'un sous-programme.

Nous reviendrons dans cette leçon sur ces deux modes d'adressage.

II.10. Instructions de transfert par registres INDEX

Nous pouvons maintenant étudier les instructions de chargement de 1 octet par registre Index.

a) LD (IX + d), n ou LD (IY + d), n
Le contenu de l'opérande «n» est chargé à l'adresse mémoire constituée par le contenu du registre index IX ou IY auquel on ajoute le déplacement «d».
Les codes binaire et hexadécimal sont :



Exemple :

LD (IX + 20 H), 05

s'écrit :

DD 36 20 05

Si IX contient 840 H, après l'exécution de cette instruction, l'emplacement mémoire

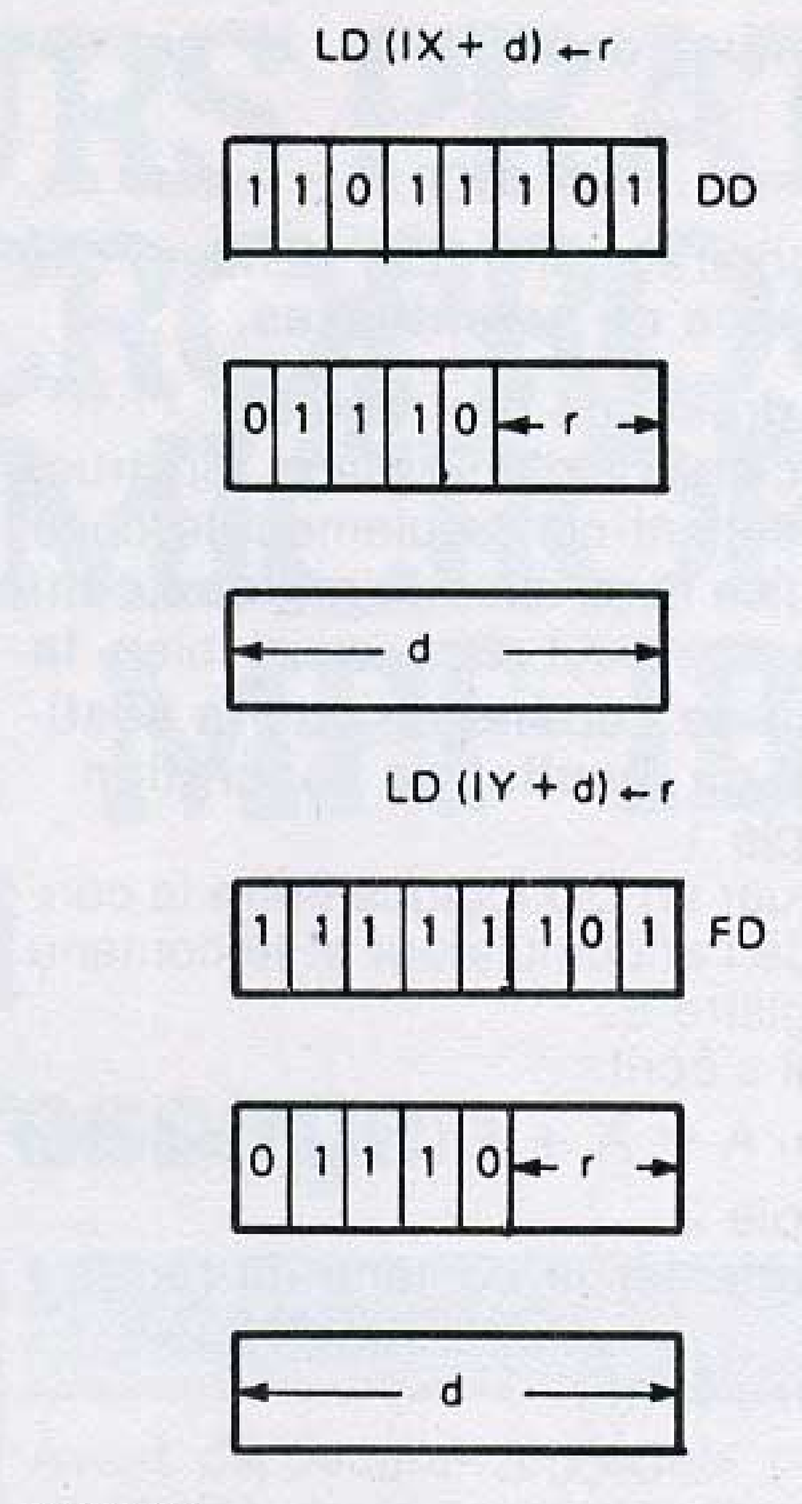
840 + 20 = 860 H,

contient la donnée 05 H.

b) LD (IX + d),r ou LD (IY + d),r
Le contenu du registre «r» spécifié dans l'instruction est chargé à l'adresse mémoire constituée par le contenu du registre index IX ou IY auquel on ajoute le déplacement «d».
Les codes binaire et hexadécimal sont : (voir page suivante)
Exemple :

LD (IY + 10 H), D

s'écrit FD 72 10.



avec r :

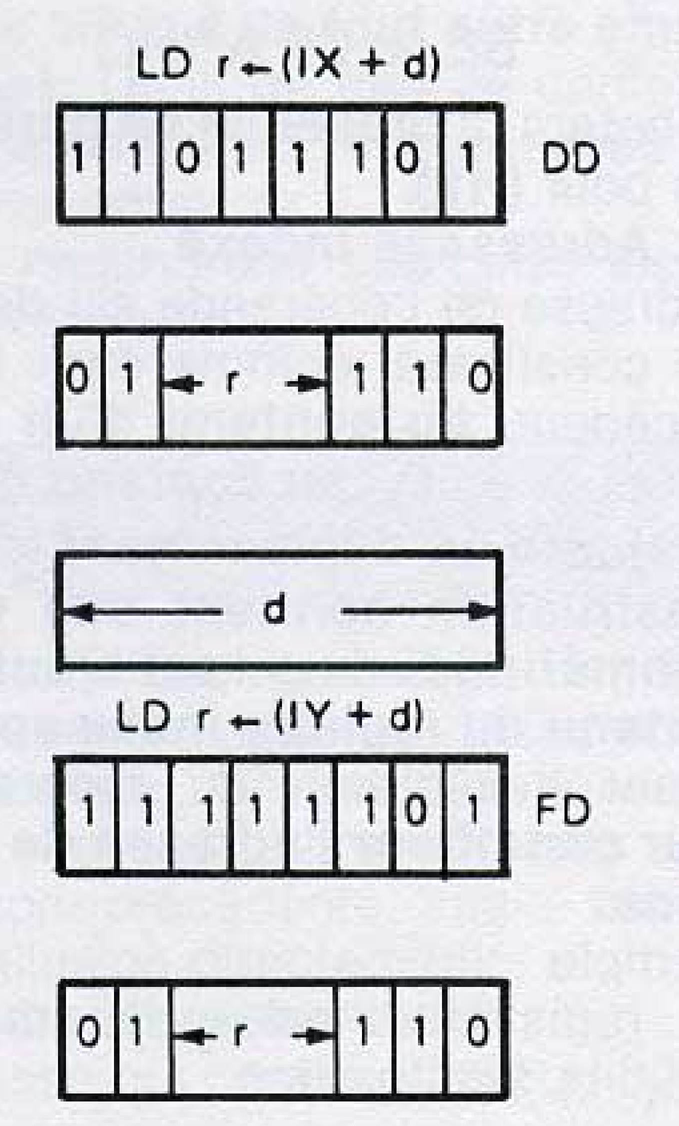
A = 111	D = 010	L = 101
B = 000	E = 011	
C = 001	H = 100	

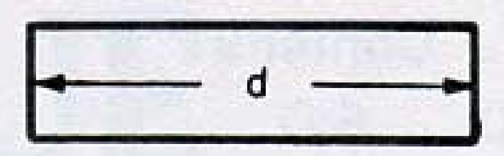
Si IY contient 21 00 H, après exécution de cette instruction, l'emplacement mémoire

21 00 + 10 = 21 10 H

est chargé par le contenu du registre D.

c) LD r, (IX + d) ou LD r, (IY + d)
Le contenu de la mémoire d'adresse constituée par le contenu du registre index IX ou IY auquel on ajoute le déplacement «d» est transféré dans le registre «r» désigné par l'instruction.
Les codes binaire et hexadécimal sont :





Exemple : LD L←(IX + 03), s'écrit DD 6E 03
Si IX contient 18 000 H, après exécution de cette instruction, l'emplacement mémoire

18 00 + 03 = 18 03 H

est chargé dans le registre «L».

II.10. Exercices

1. A quel mode d'adressage appartiennent les instructions suivantes :

a) A←A + B + C
b) A←A − D
c) HL←HL + BC + C
d) A←A (HL)
e) IX←IX + SP
f) Incrém. (IX + d)
g) EX (SP), HL
h) Load HL, (nn)
i) Load IX, nn
j) Load E, (IY + d)

2. Donner le code machine des instructions suivantes :

a) LD (IX + 3E), 50 d
b) Charger B avec le contenu adresse par (IY + 10 H)
c) Transférer le contenu de D dans (IX + 3 F)
d) LD A, (IY + 7)
e) Charger (IX + d) avec 95 d

III. INSTRUCTIONS DE SAUT, D'APPEL ET DE RETOUR

III.1. Introduction

En guise d'introduction de ces nouvelles instructions, nous écrirons un petit programme pour bien comprendre le problème tel qu'il se pose.
Il s'agit d'inscrire FF H dans les 16 cases mémoires de la RAM à partir de l'adresse 18 00 H à 18 0F H. L'écriture de ce programme commençant à l'adresse 18 50 H.
(Rappel : à la mise sous tension, le contenu d'une mémoire vive est a priori quelconque).
En examinant attentivement le répertoire d'instruction du Z 80, quoi que fort abondant, nous n'y trouvons pas une instruction qui puisse «charger directement une case d'adresse donnée par un octet». Une telle instruction s'écrirait :

LD (nn), n

ou charger la case d'adresse nn par la quantité «n».

Par contre, nous pouvons charger une case donnée «nn» par le contenu de l'accumulateur A :

Load (nn), A ou 32 nn

et charger A par une donnée FF :

Load A, FF ou 3 E FF

D'où l'écriture du programme «PO».

Adresse	Mnémonique	Codes hexadécimaux
1850	LD A, FF	3E FF
1852	LD (1800), A	3A 00 18
1855	LD (1801), A	3A 01 18
1858	LD (1802), A	3A 02 18
....		
187C	LD (180E), A	3A 0E 18
187F	LD (180F), A	3A 0F 18
1882	Halt (FIN)	76

L'écriture de ce programme ne comporte pas moins de 50 instructions (1882 H − 1850 H = 32 H soit 50 d) pour charger 16 cases mémoires avec FF ! Heureusement qu'il ne faut pas ainsi remplir les 2 K (2048) octets de la RAM !

Dans une première étape, nous allons mettre à profit ce que nous venons d'étudier à propos de l'adressage pour réduire la longueur de notre programme.

Réécrivons le programme en utilisant l'adressage par la paire de registres HL.

Le registre A contient toujours la donnée FF, mais le transfert de la donnée s'effectue dans la case mémoire pointée par la paire de registres HL.

Il suffit d'initialiser HL en chargeant ce registre double avec l'adresse de départ, et après chaque opération de transfert d'incrémenter son contenu.

On obtient ainsi : Programme P1

Programme P1

Adresse	Mnémonique	Codes hexadécimaux
1850	LD A, FF	3E FF
1852	LD HL, 1800	21 00 18 (HL) = 1800
1855	LD (HL), A	77
1856	INC HL	23 (HL) = 1801
1857	LD (HL), A	77
1858	INC HL	23 (HL) = 1802
......		
1871	LD (HL), A	77
1872	INC HL	23 (HL) = 180F
1873	LD (HL), A	77
1874	Halt (FIN)	76

Pour obtenir le même résultat qu'avec le programme PO, nous n'avons eu à utiliser que 24 H soit 36 instructions au lieu de 50. C'est un peu mieux.

En examinant attentivement la deuxième version du programme, nous remarquons que nous avons répété 16 fois le même couple d'instructions :

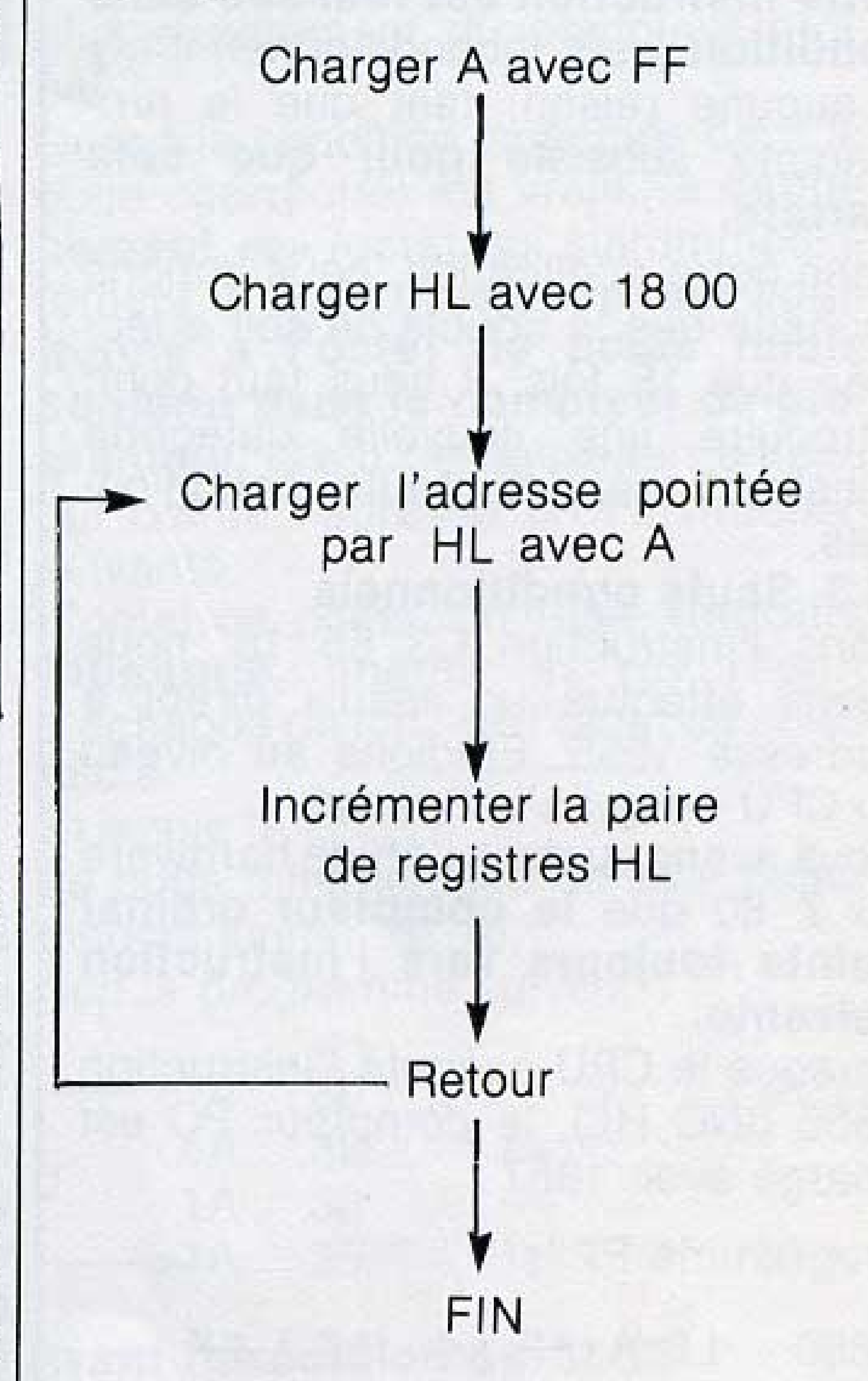
LD (HL), A et INC HL

d'où l'idée de chercher si dans le répertoire instructions du Z 80, il n'existe pas une instruction ayant pour objet : «Recommencer à exécuter la dernière ou les n dernières instructions».

Une telle instruction n'existe pas, ou tout au moins sous cette forme. Par contre, il existe tout un ensemble d'instructions qui permettent d'effectuer des «sauts» aussi bien en arrière qu'en avant. C'est ce que nous étudierons maintenant.

III.2. «Instructions de saut»

Reprenons le programme P1 et présentons-le sous la forme d'un organigramme.



Ce qui conduit au programme suivant : P2

Programme P2

Adresse	Mnémonique	Codes hexadécimaux
1850	LD A, FF	3E FF
1852	LD HL, 1800	21 00 18
1855	LD (HL), A	77
1856	INC HL	23
1857	Retour à 1855	C3 55 18
185A	Halt (FIN)	

Nous venons ainsi d'introduire une nouvelle instruction «Saut inconditionnel» à une adresse donnée. Le code opératoire est C3 suivi des deux octets d'adresse à laquelle le programme doit «sauter».

Comme d'habitude, **l'octet de poids faible suit le code opératoire et vient ensuite l'octet de poids fort.**

Ainsi, d'un programme de 50 instructions puis de 36, nous obtenons maintenant un programme de... 10 instructions.

Que le lecteur ne se précipite pas sur son MPF-1B s'il en possède un, pour essayer ce dernier programme.

L'objectif ne sera-t-il pas atteint ? Bien sûr que oui, mais comment le

programme s'arrêtera-t-il ? Car a priori l'instruction «Halt» ne pourra jamais être exécutée.
En effectuant à l'adresse 1857 un retour à 1855, nous effectuons une «boucle de programmation» (ou LOOP en anglais). Etant donné que **cette instruction est réalisée sans condition** (saut inconditionnel) il n'y a aucune raison, tant que le programme subsiste **pour que cela s'arrête.**
Dans notre application, nous aurions souhaité que la boucle ne soit effectuée que 16 fois, il nous faut donc introduire une nouvelle catégorie d'instructions, les sauts conditionnels.

III.3. Sauts conditionnels

Dans l'instruction C3 55 18, nous avons effectué un «saut» direct à l'adresse 1855. Etudions au niveau du CPU ce qui s'est passé.
Nous avons vu dans l'étude hardware du Z 80 que **le compteur ordinal pointe toujours vers l'instruction suivante.**
Lorsque le CPU exécute l'instruction 1856 (INC HL), le compteur PC est chargé avec 1857.

Programme P2

1850	LD A, FF	3E	FF	
1852	LD HL, 1800	21	00	18
1855	LD (HL) A	77		
1856	INC HL	23		
1857	Retour à 1855	C3	55	18
185A	Halt (FIN)			

ou

1850	3E	Instruction 1
1851	FF	
1852	21	Instruction 2
1853	00	
1854	18	
1855	77	Instruction 3
1856	23	Instruction 4
1857	C3	Instruction 5
1858	55	
1857	18	
185A	76	Instruction 6

Après avoir effectué + 1 dans HL, le CPU lit l'instruction suivante (5) composée de 3 octets (C3, 55 et 18), puis le compteur PC s'incrémente : il contient 185A (instruction 6).
L'exécution de l'instruction (5) : (C3 55 18) charge le compteur PC avec l'adresse 1855 et par le fait même «écrase» le contenu précédent (185A) : le programme exécutera l'instruction (3) (contenue dans 1855) au lieu de l'instruction (6) (contenue dans 185A).

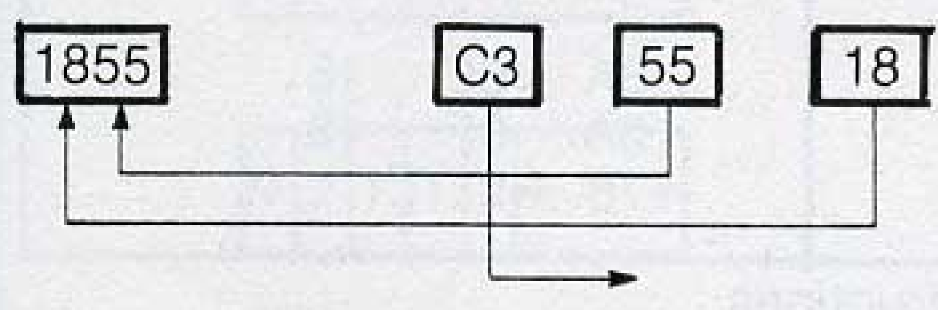
D'où le schéma :
Avant l'exécution de l'instruction (5) [1857]

PC REGISTRE INSTRUCTIONS

185A C3 55 18

Instruction lue qui va être exécutée

Après l'exécution de l'instruction (5) [1857]



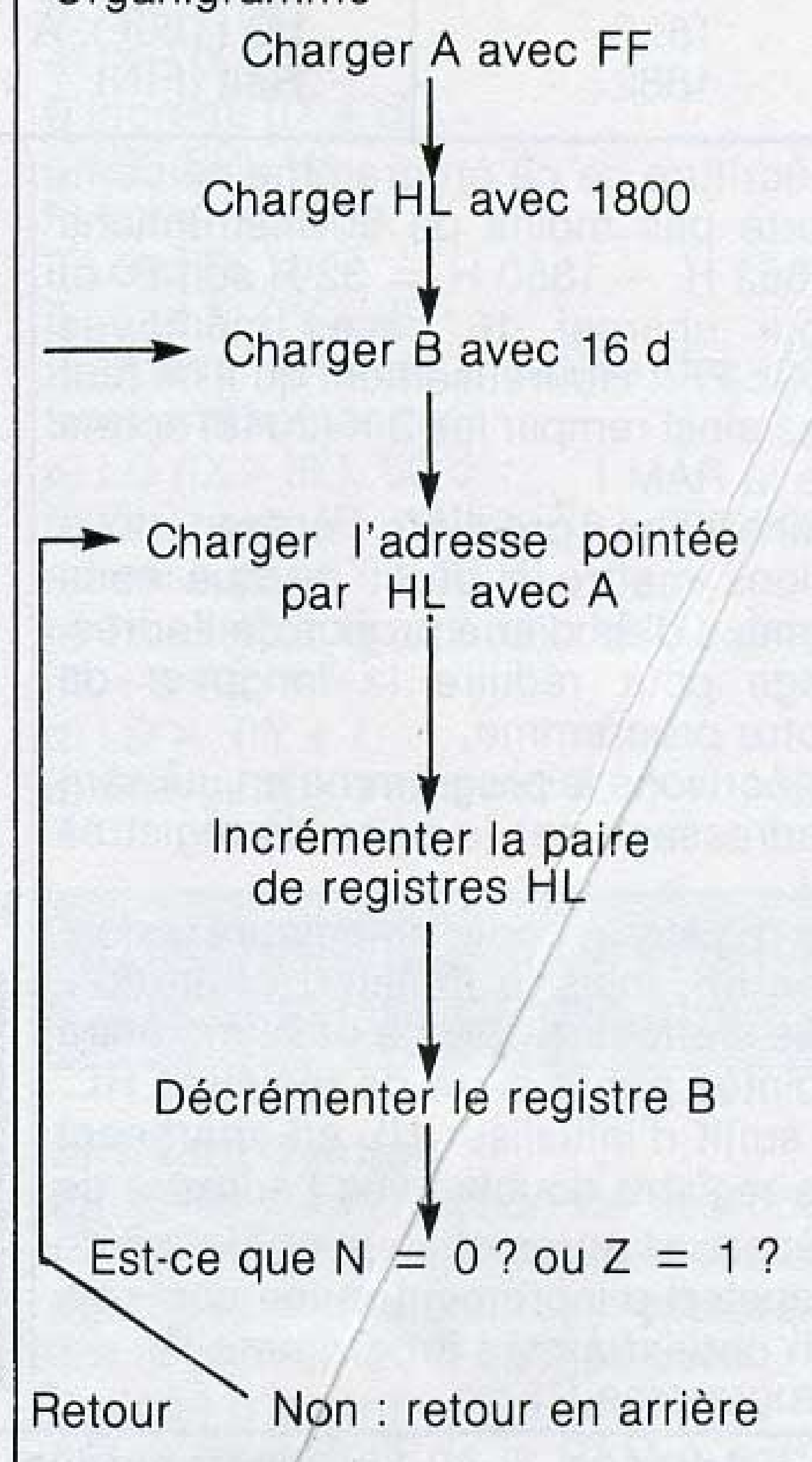
charger PC avec les 2 octets suivants

En n'introduisant aucune condition, le programme reviendra (tant qu'il subsiste) a 1855 après l'instruction 1857 : ce ne sera plus 16 cases mémoires qui contiendront FF, mais toute la mémoire vive ! En réalité, notre programme sera détruit par lui-même avant.
Nous allons donc modifier notre programme en utilisant un compteur. Nous utiliserons le registre B comme compteur.
Nous comptabiliserons dans B le nombre de fois que nous avons exécuté la «boucle» :

(LD (HL), A et IHC HL)

Quand 16 sera atteint, nous arrêterons le programme.
En réalité, il est plus aisé de détecter le «passage à zéro», aussi nous chargerons B avec 16 que nous décrémenterons (DEC B) à chaque «passage» et nous surveillerons le passage à «0».
Ainsi, tant que le contenu de B est différent de 0, nous répétons la boucle.

Quand B = 0, nous effectuons l'instruction suivante, c'est-à-dire HALT (FIN).
Reste à détecter le «passage à zéro» de B. Nous avons vu dans les instructions «INC» et «DEC» que les indicateurs (Registre F) étaient positionnés en fonction du résultat. Lorsqu'on effectue une décrémentation de B (05), l'indicateur Z sera à 1 quand le contenu de B sera nul.
En conclusion, la boucle sera effectuée tant que Z = 0 (donc B ≠ 0).
L'instruction «saut à «nn» si Z = 0 est C2.
Nous obtenons ainsi un nouveau programme et un nouvel organigramme.
Organigramme

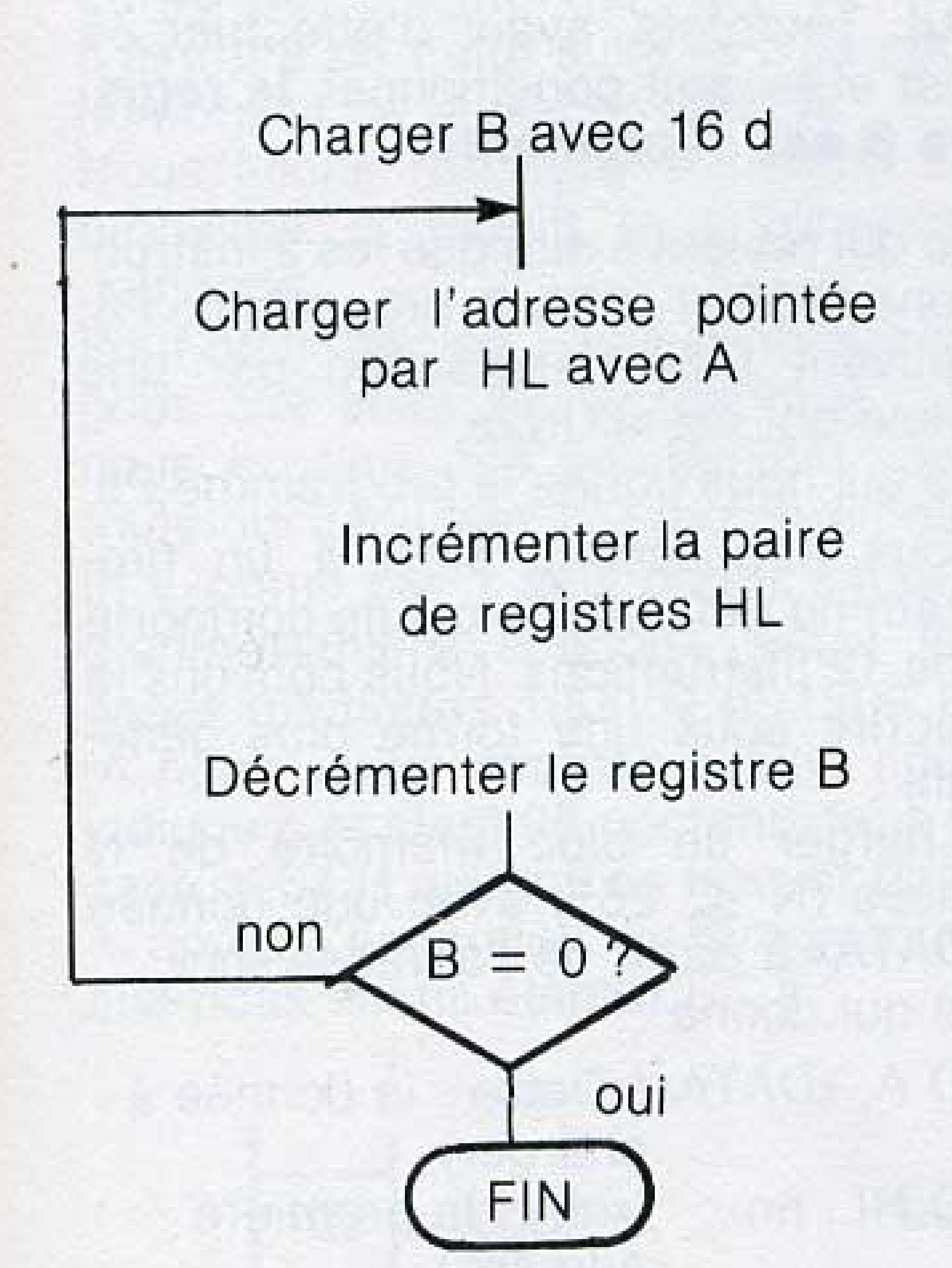


Programme P3

Adresse	Mnémonique	Codes hexadécimaux		
1850	LD A, FF	3E	FF	
1852	LD HL, 1800	21	00	18
1855	LD B, 0FH	06	0F	
1857	LD (HL), A	77		
1858	INC HL	23		
1859	DEC B	05		
185A	Saut à 1857 si Z=0	C2	57	18
185D	Halt (FIN)	76		

Notre programme P3 comporte 14 instructions et exécute ce qui est demandé en s'arrêtant sitôt l'exécution terminée : c'est exactement ce que nous souhaitons.
Nous allons présenter l'organigramme avec les éléments de programmation.





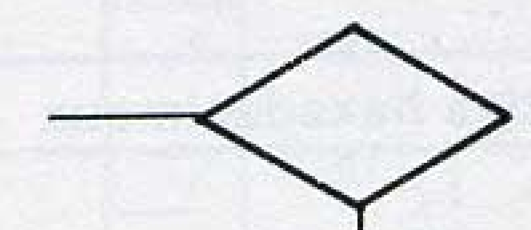
Symbole pour indiquer le début d'un programme

DEBUT

Symbole pour indiquer la fin d'un programme.
Une instruction comme Halt (76) doit terminer un programme.



Symbole de test. Il y a deux alternatives possibles et seulement deux.



Nota : Que se passe-t-il quand Z = 1 ? (ou B = 0).
Au moment de l'exécution de l'instruction 185A, le compteur ordinal PC contient 185D. Seulement au lieu d'être modifié avec 1857 (quand Z = 0), il n'y a **pas de changement de PC** : l'instruction suivante est alors 185D.

III.4. Sauts en adressage étendu immédiat

Les instructions de sauts en adressage étendu immédiat se présentent toujours sous la forme d'une instruction de 3 octets. Le premier contient la condition (qui peut être inconditionnelle), le second indique le poids faible de l'adresse de branchement et le troisième représente l'octet de poids fort.
Hormis l'instruction C3 nn, saut inconditionnel, la condition de branchement est l'état «1» ou «0» de l'un des indicateurs du registre F. Ce qui conduit au tableau suivant :

Code opératoire	Condition de saut
C3 nn	Saut inconditionnel
DA nn	Saut à nn si C = 1
D2 nn	Saut à nn si C = 0
CA nn	Saut à nn si Z = 1
C2 nn	Saut à nn si Z = 0
EA nn	Saut à nn si parité impaire
E2 nn	Saut à nn si parité paire
FA nn	Saut à nn si signe négatif
F2 nn	Saut à nn si signe positif

A noter qu'aucun indicateur n'est affecté par ces instructions.

A noter qu'aucun indicateur n'est affecté par ces instructions.

III.5. Sauts en adressage relatif

Dans ce type de branchement, il ne s'agit plus de faire un saut à une adresse quelconque du programme, mais d'effectuer **un déplacement en avant ou en arrière par rapport à l'instruction** qui détermine le branchement.
L'instruction de saut relatif est constituée de 2 octets. Le premier indique la condition requise pour que le saut ait lieu (le «saut» peut être inconditionnel). Le second indique la **valeur algébrique** du déplacement.
Comme la valeur du déplacement est un «nombre signé», si le bit 7 (poids le plus fort) est «0», le déplacement est positif. Si le bit 7 est «1», le déplacement est négatif, c'est un retour en arrière.
Ainsi, le déplacement e est tel que :

$$-128 \leqslant e \leqslant +127 \text{ (décimal)}$$
$$1111\ 1111 \leqslant e \leqslant 0111\ 1111$$
(en binaire)
$$FF \leqslant e \leqslant 7F$$
(en hexadécimal)

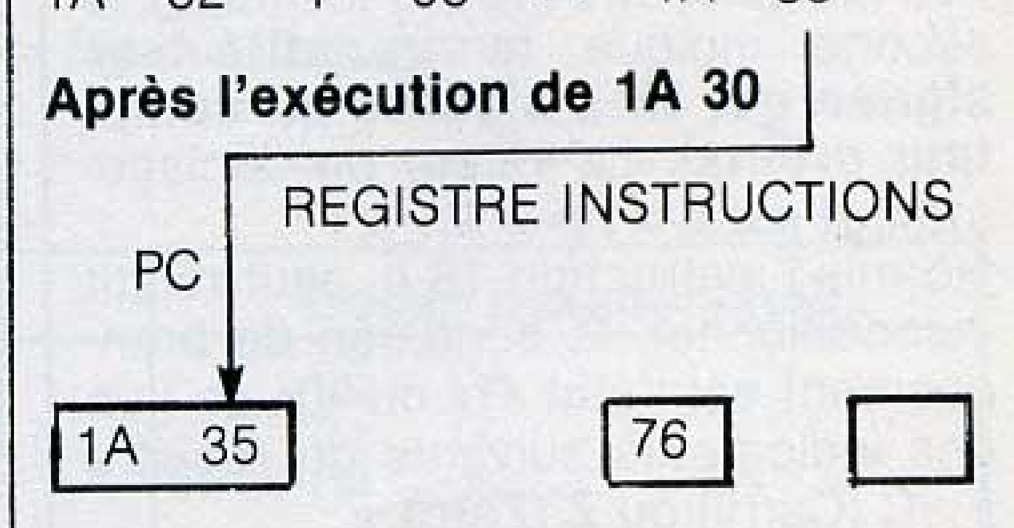
Que se passe-t-il au niveau du CPU et plus exactement du compteur ordinal ?
Quand la condition (spécifiée dans le code opératoire) est vraie, le **déplacement «e»** (octet qui suit immédiatement le code opératoire) **est additionné à l'octet de poids faible contenu dans le compteur de programme.** C'est ce nouveau contenu qui pointe l'adresse de l'instruction suivante.
L'octet de poids fort reste **toujours inchangé,** même si un report s'échappe du bit 7 de l'octet de poids faible.
Exemple 1
Le code mnémonique du saut relatif inconditionnel est 18.
Soit le programme suivant :

1A	30	JR + 3	18 03
d	1A	32	—	
	1A	33	—	
	1A	35	Halt	76

Avant l'exécution de 1A 30

REGISTRE INSTRUCTIONS
PC
1A 32 — 18 — 03
Instruction lue qui va être exécutée
1A 32 + 03 = 1A 35

Après l'exécution de 1A 30



La quantité 03 est additionnée à l'octet de poids faible du compteur PC

La prochaine instruction qui va être exécutée est celle contenue dans l'emplacement 1A 35.
La valeur du déplacement

d = 1A 35 − 1A 30 = 5 ou d = e + 2

Adresse	Mnémonique	Codes hexadécimaux
1850	LD A, FF	3E FF
1852	LD HL, 1800	21 00 18
1855	LD (HL), A	77
1856	INC HL	23
1857	Retour «d»	18 e
1859	Halt (FIN)	76

Exemple 2
Reprenons le programme P2 dans lequel nous remplaçons le saut inconditionnel en adressage immédiat étendu par un saut en adressage relatif.

Au moment de l'exécution de l'instruction 1857 (ici il n'y a que 2 octets), le contenu de PC est 1859. Pour que la prochaine instruction soit 1856, il faut retrancher 4 au contenu de PC, soit additionner (−4) ou en valeur hexadécimale signée FCH.
En effet :

```
        59 H
        FC H
(1)     54 H
```

Le report qui s'échappe de l'octet de poids faible (1) est perdu.
L'adresse de l'instruction suivante est 1854 H.
Le déplacement

$$d = e + 2 = -4 + 2 = -2$$

En effet, à partir de l'instruction 1857, le programme revient à 2 instructions antérieures.

III.6. Instructions de sauts en adressage relatif

Les instructions de sauts en adressage relatif se présentent toujours sous la forme d'une instruction de 2 octets. Le premier précise la condition (qui peut être inconditionnelle), le second indique **la quantité «e» signée qui sera ajoutée au compteur ordinal.** La valeur du déplacement d est d = «e» + 2.
Hormis l'instruction 18 d, saut relatif inconditionnel, la condition de branchement est l'état «1» ou «0» de l'un des indicateurs suivants du registre F : C (Carry) ou Z (Zéro).
Ce qui conduit au tableau suivant :

Code opératoire	Condition de saut
18 d	Saut inconditionnel
38 d	Saut à PC + d si C = 1
30 d	Saut à PC + d si C = 0
28 d	Saut à PC + d si Z = 1
20 d	Saut à PC + d si Z = 0

III.7. Comparaison entre les sauts en adressage immédiat et relatif

Avantages du saut en adressage relatif :
- l'instruction ne comporte que 2 octets
- le programme est «transposable».

Le programme précédent de l'exemple 2, qui est stocké arbitrairement à partir de l'adresse 1850, pourrait tout aussi bien être écrit à partir de 1950, sans changer quoi que ce soit à son contenu. Ce qui n'est pas le cas dans l'adressage immédiat puisque l'adresse de branchement est précisée (saut à 1857).
Inconvénients :
- le champ de branchement est limité

$$-129 \leqslant d \leqslant +127$$

- les indicateurs P et S ne peuvent intervenir comme condition.

III.8. Instructions de saut en adressage par registres

Il existe un ensemble de trois instructions qui ne permettent **d'effectuer que des sauts inconditionnels** à l'adresse dans **l'une des paires de registres** HL, IX ou IY.
Les codes mnémoniques sont :
E9 PC←HL : le compteur ordinal est chargé par le contenu de la paire de registres HL
DD E9 PC←IX : le compteur ordinal est chargé par le contenu du registre ou FD E9 PC←IY : Index IX ou IY.

III.9. Instruction «DJNZ»

Reprenons le programme P3 que nous allons modifier en remplaçant le branchement par adressage immédiat par un branchement en adressage relatif, nous obtenons P4.

L'instruction «DJNZ» (dont le mnémonique est 10 e) effectue un saut relatif comme l'instruction 20, mais à la **condition que le contenu du registre B** (et uniquement lui) **ne soit pas nul.** Toutefois, avant d'effectuer ce test et le saut conditionnel, **le registre B est décrémenté.**

Ce qui revient à dire que les 2 instructions 1859 et 185A du programme P4, peuvent être remplacées par une seule qui est «DJNZ».
Ce qui nous donne le programme P5.
Nous aboutissons ainsi à un programme définitif P5, qui ne comporte que 12 instructions. Nous pouvons le décrire sous une forme plus générale :
«Charger un bloc mémoire de N cases (N ⩽ 255) avec une donnée «DATA» à partir de l'adresse «nn».
Ce qui donne :

LD A, «DATA» («Data» : la donnée à charger)
LD HL, nn («nn» : la première adresse)

Programme P4

Adresse	Mnémoniques	Codes hexadécimaux
1850	LD A, FF	3E FF
1852	LD HL, 1800	21 00 18
1855	LD B, 0F H	06 0F
1857	LD (HL), A	77
1858	INC HL	23
1859	DEC B	05
185A	JR NZ, −3	20 FB
185C	Halt (FIN)	76

Programme P5

Adresse	Mnémoniques	Codes hexadécimaux
1850	LD A, FF	3E FF
1852	LD HL, 1800	21 00 18
1855	LD B, 0F	06 0F
1857	LD (HL), A	77
1858	HL, HL + 1	23
1859	DJNZ, −2	10 FC
185B	Halt (FIN)	76

LD B, N («N» : le nombre de cases mémoires)

LD (HL), A
INC HL
DJNZ, – 2
Halt (FIN)

III.10. Exemple d'application

Les connaissances ainsi acquises sont largement suffisantes pour résoudre des problèmes concrets.

Problème

Soit une suite de 10 nombres positifs (exprimés en hexadécimal sur 1 seul octet) rangés dans un ordre quelconque dans les adresses 1900 H à 1909 H.

Ecrire le programme qui permet d'extraire de cette suite le plus petit nombre et de le placer dans A.

Nous allons résoudre ensemble ce problème et établir le programme demandé.

Pour bien fixer les idées, établissons-nous une suite de 10 nombres que nous exprimerons en décimal sans toutefois dépasser 127 (nombre positif exprimé par 1 seul octet), rangée d'une manière aléatoire entre les cases mémoires 1900 H et 1909 H. (A noter qu'un examen rapide nous indique que c'est 08 contenu dans la case 1904 H qui doit se trouver dans A après l'exécution du programme que nous allons établir).

	18FF
109	1900
76	1901
89	1902
49	1903
08	1904
17	1905
120	1906
29	1907
33	1908
12	1909
	190A

Le principe adopté est de comparer le contenu du registre B avec successivement chacun des nombres N de la suite.

Si le résultat est négatif (N > B), le registre B contient un nombre plus petit que N, on ne change rien dans B.

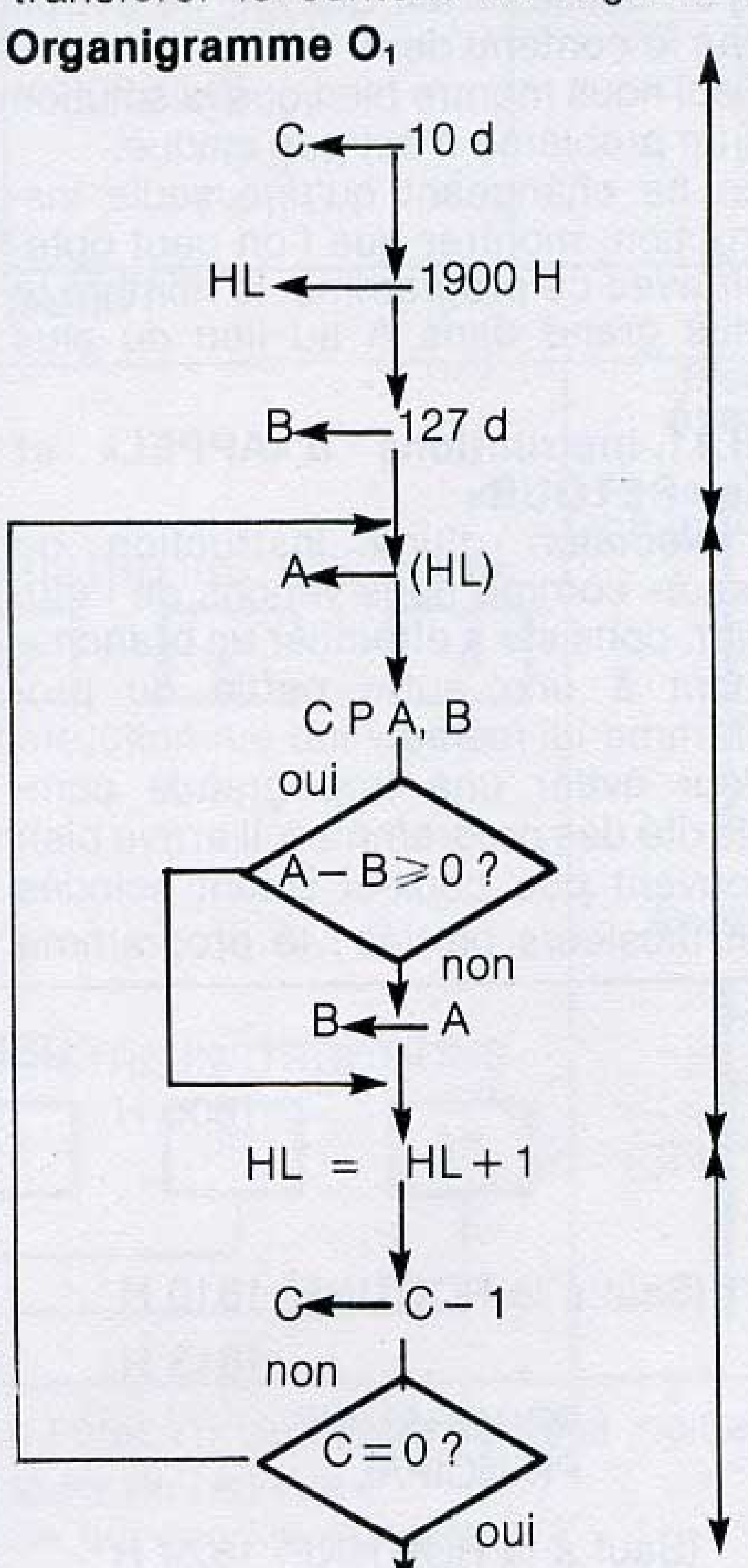
Si le résultat est positif (N < B), N est plus petit que B et l'on place N dans B.

Lorsque les 10 nombres auront ainsi été examinés, il ne restera plus qu'à transférer le contenu du registre B dans A.

Nous allons suggérer un premier organigramme basé sur ce principe. Nous ne saurions trop insister pour que le lecteur établisse lui-même le programme en «code machine» correspondant et l'exécute avec son MPF-1B.

Le programme est écrit à partir de l'adresse 1800 H.

Organigramme O_1



Initialisation :
nombre de chiffres, adresse de départ et valeur maximale

Boucle (Loop) de comparaison
si A ⩾ B, B inchangé
si A < B, B chargé avec A

Recherche du chiffre N suivant avec test du dernier

Transfert du résultat dans A

Fin du programme

Ecriture du programme Pap 1 :

Adresse	Mnémoniques	Codes hexadécimaux		
1800	LD C, 10 d	0E	0A	
1802	LD HL, 1900 H	21	00	19
1805	LD B, 127 d	06	7F	
1807	LD A, (HL)	7E		
1808	CP A, B	B8		
1809	JP, S	F2	0D	18
180C	LD B, A	47		
180D	INC HL	23		
180E	DEC C	0D		
180F	JR, Z	20	F6	
1811	LD A, B	78		
1812	Halt (FIN)	76		

Questions :

a) Le programme ainsi écrit est-il transposable ?

b) Quelles suggestions peut-on faire pour le simplifer ? (Notamment pour réduire le nombre d'instructions)

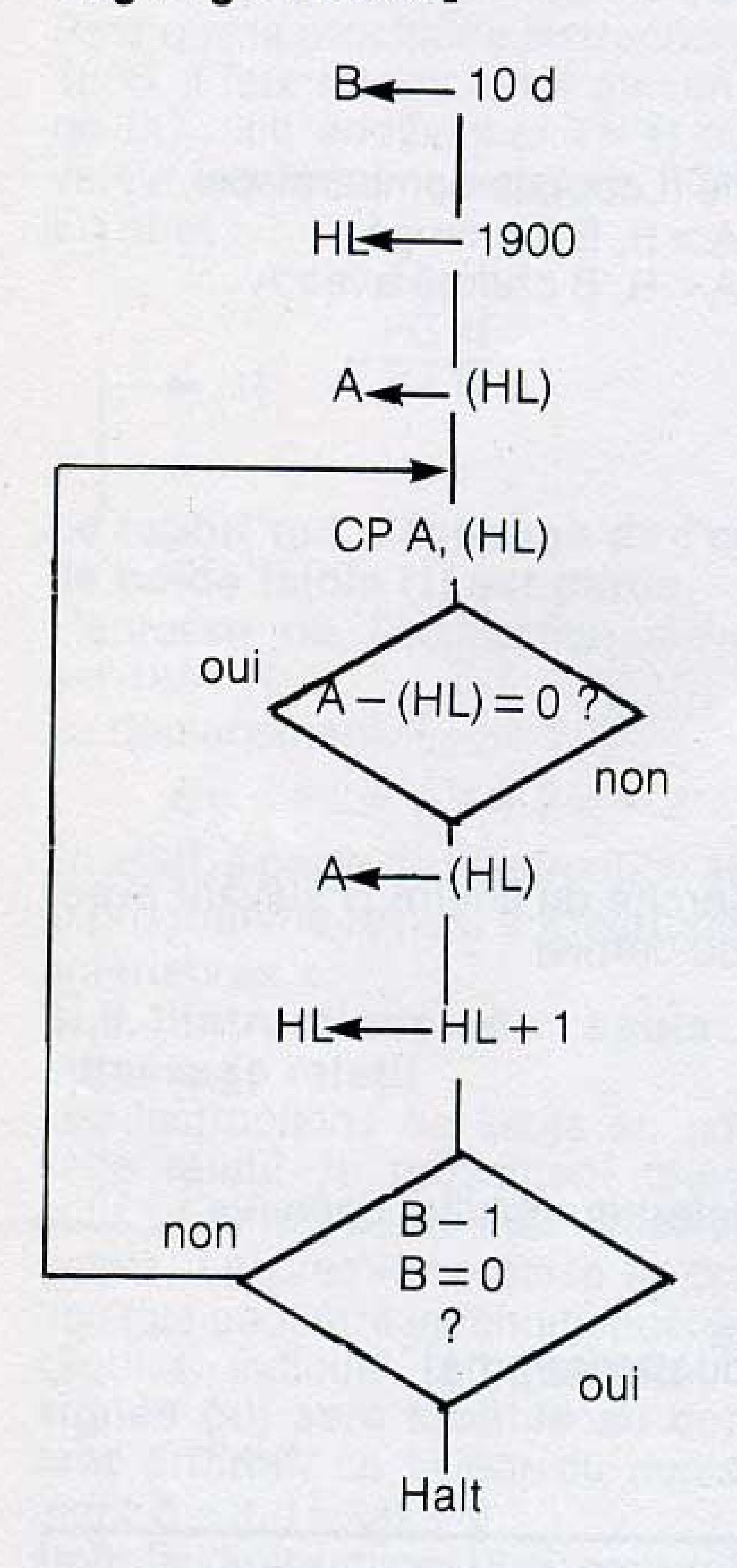
c) En modifiant l'organigramme de départ, établir un autre programme (plus simple si possible)

Nota :

Si des erreurs sont apparues dans l'élaboration du programme, nous vous conseillons de vous reporter à l'étude de l'instruction correspondante.

Nous présentons ici une seconde solution :

Organigramme O_2



Programme Pap 2

Adresse	Mnémoniques	Codes hexadécimaux
1800	LD B, 10 d	06 0A
02	LD HL, 1900	21 00 19
05	LD A, (HL)	7E
Loop 06	CP A, (HL)	BE
07	JP,	FA 0B 18
0A	LD A, (HL)	7E
0B	INC HL	23
0C	DJNZ, (Loop)	10 F8
0E	Halt (FIN)	76

Nous obtenons un programme de 15 instructions au lieu de 19.

Les remarques que l'on peut faire sont :

a) on considère la première valeur comme la plus petite

b) on utilise l'adressage par HL pour la comparaison

c) on utilise DJNZ, pour le comptage

d) on utilise le fait que CP ne détruit pas le contenu de A.

Ceci nous montre bien que la solution à un problème n'est pas unique.

En ne changeant qu'une seule instruction, montrer que l'on peut obtenir avec ce programme, le nombre le plus grand dans A au lieu du plus petit.

III.11. Instructions d'«APPEL» et de «RETOUR»

L'exécution d'une instruction de «saut» comme nous venons de l'étudier, consiste à effectuer un branchement à une autre partie du programme lui-même.

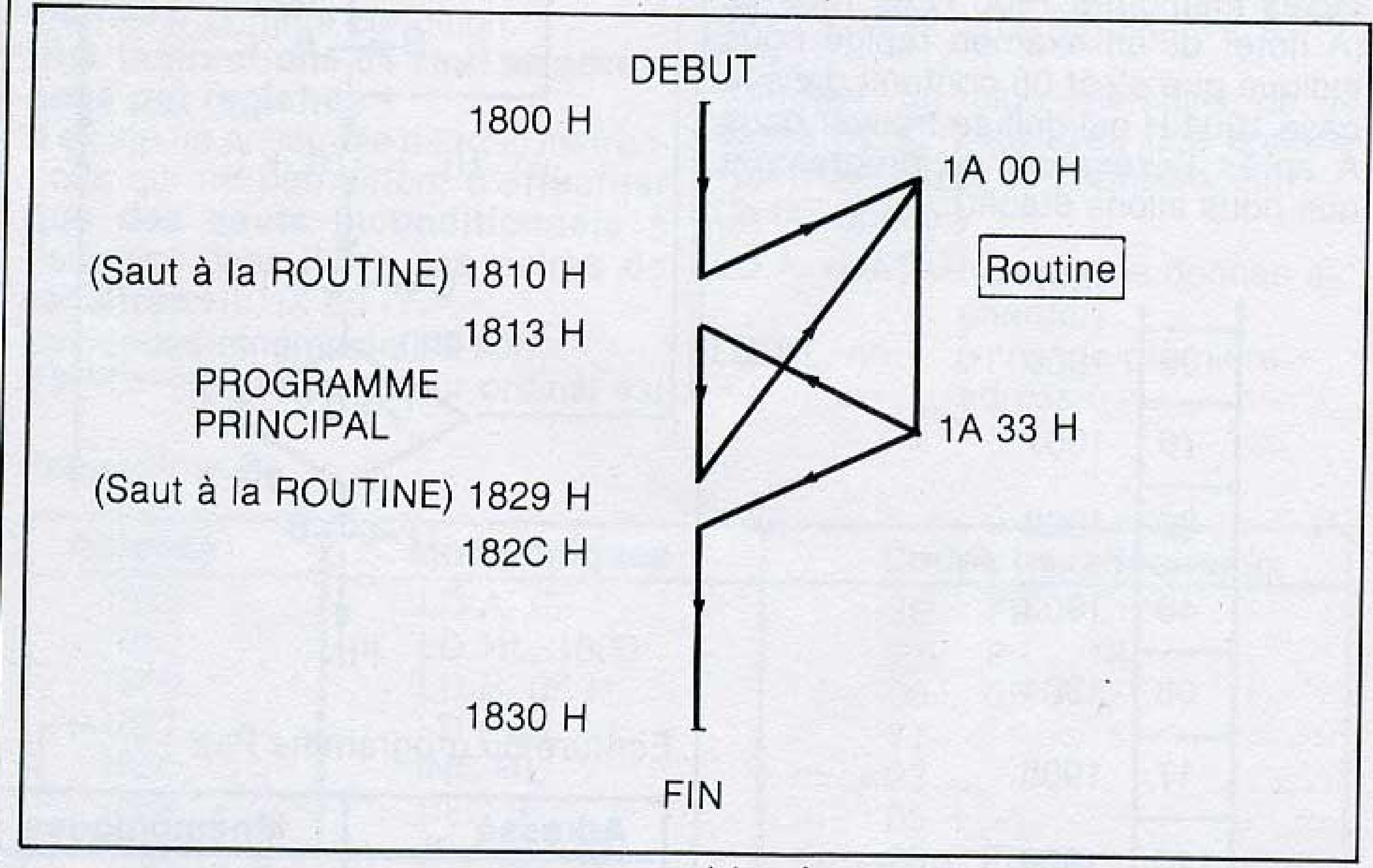
Pour éviter une trop grande complexité des programmes, il arrive bien souvent que ceux-ci soient scindés en plusieurs parties : le programme principal par lui-même et un ou plusieurs sous-programmes.

Un «sous-programme» est un programme par lui-même, mais qui effectue une (ou un ensemble de) tâche bien précise, à laquelle le programme principal est souvent amené à faire «APPEL».

Par exemple, si le programme principal doit effectuer des opérations arithmétiques sur des quantités introduites en BCD, il faudra avant tout les convertir en «binaire». De même le résultat qui apparaîtra en «binaire» devra être transcodé en «BCD».

Il est bien évident que les deux séquences de transcodage peuvent être introduites dans le programme principal. Seulement chaque fois que le programmeur voudra les utiliser, il devra les réintroduire intégralement. Ceci risque fort d'alourdir sérieusement son travail.

On préfère de beaucoup utiliser des «sous-programmes» ou «routines» que l'utilisateur peut placer là où il veut dans le champ adressable de la mémoire. Généralement, on les place avant ou après le programme principal.



L'exécution d'une instruction d'«APPEL» **suspend momentanément le déroulement du programme** en cours, qui sera repris sitôt la routine exécutée.

La figure 5 schématise le déroulement d'un programme avec deux branchements à la routine qui commence en 1A00 pour se terminer en 1A33.

Supposons que l'instruction soit «APPEL inconditionnel», le code opératoire est analogue à l'instruction de

branchement en adressage immédiat. Le code héxadécimal est D3 suivi de la première adresse de la routine.
Ainsi, nous aurons en 1810 H :

D3 00 1A

Comme il s'agit d'une instruction de 3 bytes, avant l'exécution de 1810 H, le contenu du compteur ordinal sera

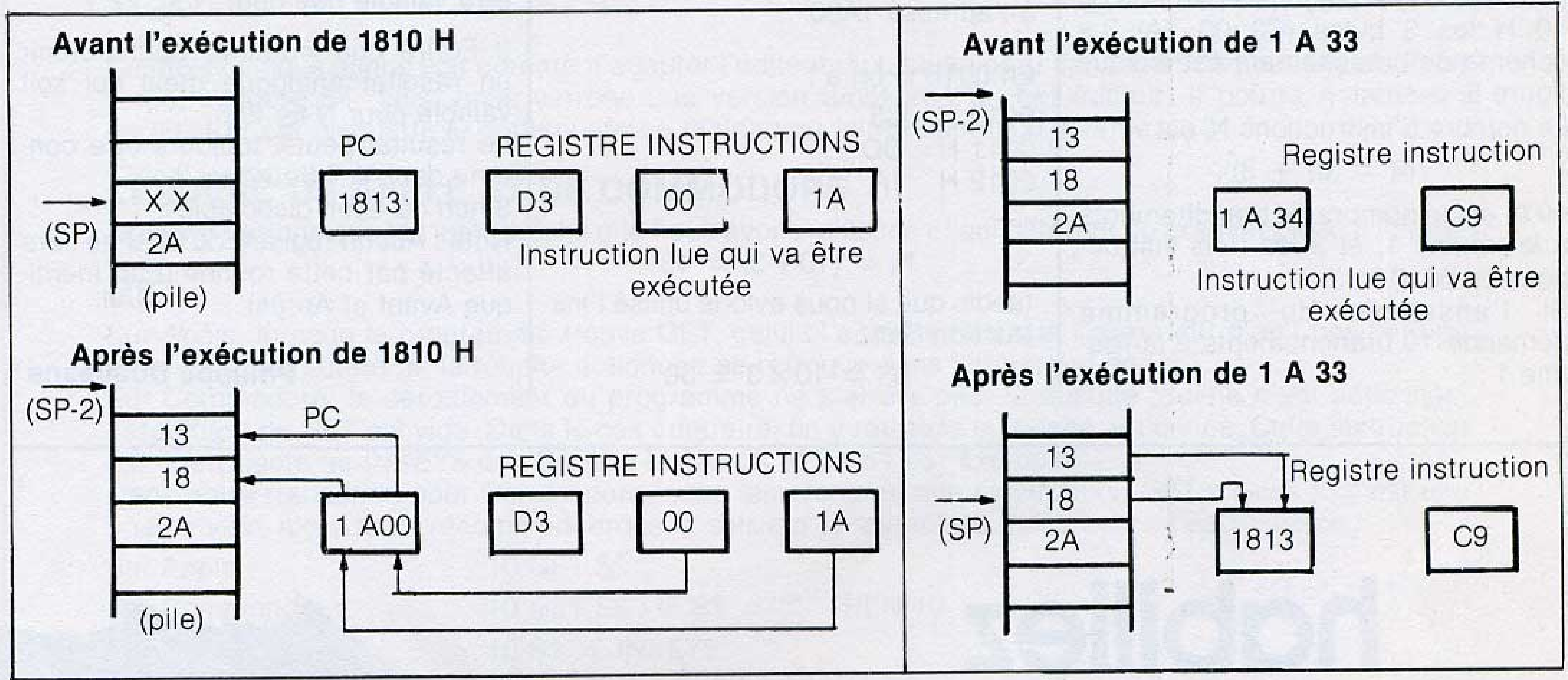
1813 H (1810 H + 3)

Après l'exécution de la routine, le programme principal doit être repris, là même où il a été suspendu.
D'où le schéma de l'exécution de

D3 00 1A.

CODE OPERATOIRE		CONDITION
APPEL	RETOUR	
CD n n	C9	Inconditionnel
DC n n	D8	Si C = 1
D4 n n	D0	Si C = 0
CC n n	C8	Si Z = 1
C4 n n	C0	Si Z = 0
EC n n	E8	Si parité impaire
E4 n n	E0	Si parité paire
FC n n	F8	Si signe négatif S = 1
F4 n n	F0	Si signe positif S = 0

Tableau I



Avant de charger PC avec le contenu de l'opérande (1A00), le contenu du compteur ordinal est sauvegardé dans la pile. Le registre pointeur de pile SP est décrémenté de 2.
Le programme saute à 1A00 H pour exécuter la routine. Celle-ci doit impérativement se terminer par une instruction de retour. Ainsi, si en 1A33, fin de la routine, nous avons une instruction de retour (sous-entendu) au programme suspendu, le schéma d'exécution de cette instruction est représenté ci-dessous.
Le code hexadécimal de retour inconditionnel est C9.
L'instruction de retour charge le compteur ordinal avec l'adresse du programme suspendu qui avait été sauvegardée dans la pile.
Les instructions d'APPEL sont toujours effectuées en adressage immédiat étendu (2 octets).
Les instructions d'APPEL et de RETOUR peuvent être conditionnées à l'état «1» ou «0» de l'un des indicateurs du registre F.
Ce qui conduit au tableau I.

III.12. Instruction «RESTART»

Nous venons d'étudier les instructions d'APPEL qui nécessitent 3 bytes dus essentiellement au fait qu'il s'agit d'un adressage immédiat étendu.

Certains microprocesseurs et notamment le Z 80 possèdent des instructions dites «restart» d'un seul octet qui permettent d'effectuer un saut inconditionnel avec sauvegarde du contenu de PC dans la pile à l'une des huit adresses comme il l'est indiqué sur le tableau II.
Pendant l'exécution d'une **instruction restart,** le contenu du **compteur ordinal est d'abord sauvegardé dans la pile.**
Le PC est chargé pour l'octet de poids fort avec «OOH» et pour l'octet

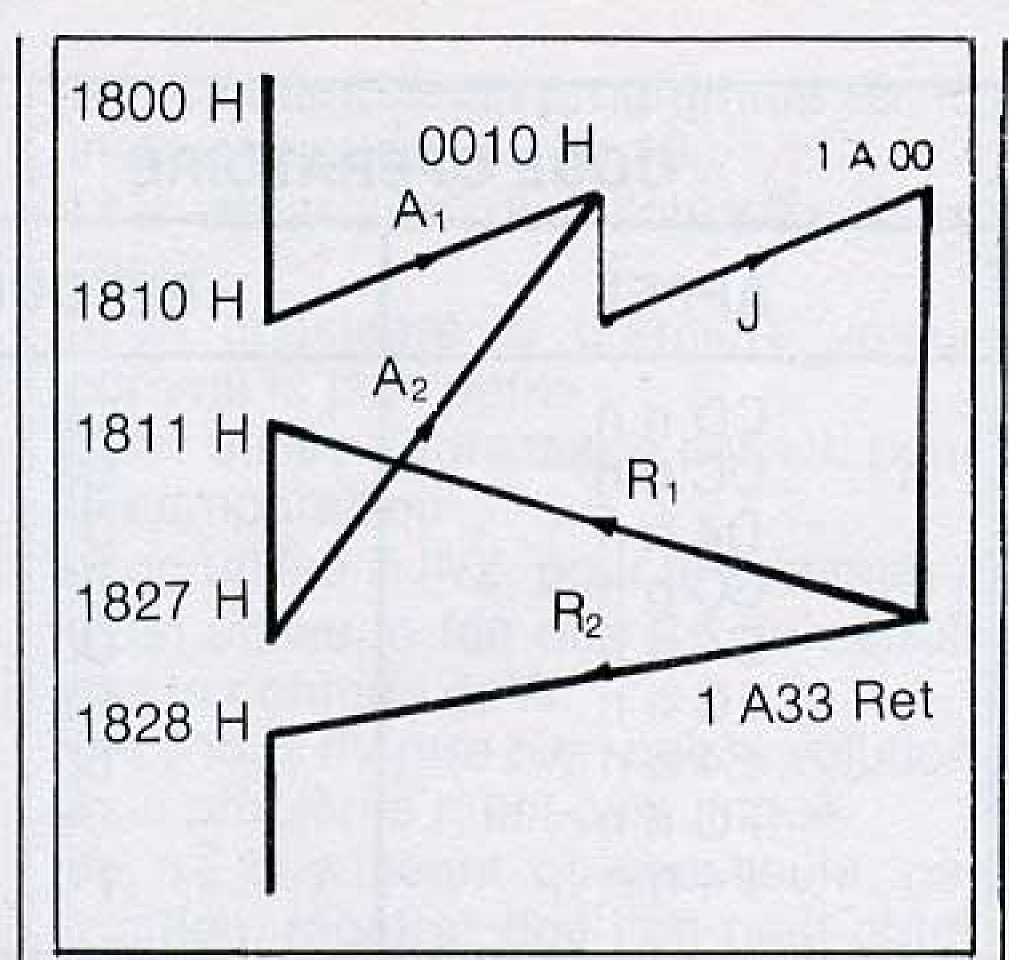
Mnémonique	Code opération	Adresse
RST «0»	C7	00 00 H
RST «8»	CF	00 08 H
RST «16»	D7	00 10 H
RST «24»	DF	00 18 H
RST «32»	E7	00 20 H
RST «40»	EF	00 28 H
RST «48»	F7	00 30 H
RST «56»	FF	00 38 H

Tableau II

de poids faible l'un des octets comme l'indique le tableau en fonction du code opération.
Exemple :
Supposons que le programme principal effectué au cours de son déroulement des Appels à une routine écrite à partir de 1A00.
Nous avons étudié dans le paragraphe précédent comment l'instruction CALL (CD 00 1A) était utilisée dans ce cas. Chaque fois qu'il sera nécessaire de faire appel à la routine, nous devons utiliser 3 bytes dans le programme.
Remplaçons l'instruction CALL du programme principal par Restart (RST 16 : D7) et plaçons à partir de 00 10 H les 3 bytes (C3 00 1A). Le schéma de branchement est indiqué figure 8.
Le nombre d'instructions N_i est :

$$N_i = (b_1 + 3)$$

où b_1 est le nombre de branchements à la routine 1, et 3 les trois instructions en 0010 H.
Si l'ensemble du programme demande 10 branchements à la routine 1 :



Saut inconditionnel :
à l'adresse 1A00

En 0010 H on a :
0010 H C3
0011 H OO
0012 H 1A

$$N_i = (10 + 3) = 13$$

tandis que si nous avions utilisé l'instruction Call :

$$N'_i = 10 \times 3 = 30$$

Dans cet exemple, le gain est de 17 instructions.

A noter qu'en 0010 H, l'instruction est un branchement inconditionnel et non pas un «Appel».

III.13. Exercices

Nota : Prendre comme première adresse de vos programmes 1800 H

1. Ecrire un programme simple qui additionne les N premiers nombres entiers et place le résultat dans l'accumulateur.
Prendre N = 12.
Pour N = 18, que faut-il faire ?
Pourquoi ce programme ne peut-il être valable que pour N $\leqslant$ 22 ?

2. Ecrire une «routine» pour obtenir un résultat analogue mais qui soit valable pour N $\leqslant$ 255.
Le résultat peut-il toujours être contenu dans A ? Pourquoi ?
Sinon où est-il disponible ?
Nota : Aucun registre du CPU ne sera affecté par cette routine (Etat identique Avant et Après).

Philippe Duquesne

COURS PRATIQUE AVEC LE MICROPROFESSOR MPF-IB

NEUVIEME PARTIE Contrôle du CPU et E/S

SOMMAIRE

I. INTRODUCTION

Avec la neuvième partie, la phase théorique du cours «Microprocesseurs» se termine. Notre objectif est de balayer l'ensemble du répertoire des instructions du Z-80 et de montrer à l'aide de quelques exemples leur mise en œuvre.

Dans les prochains numéros, nous nous consacrerons essentiellement à des applications, que chacun d'entre vous pourra exécuter sur son matériel, le Microprofessor MPF-IB. Chacune d'elles vous permettra de bien comprendre le déroulement d'un programme, d'en suivre son évolution et peu à peu de concevoir vos propres applications.

II. INSTRUCTIONS DE «DECALAGE» ET «ROTATION»

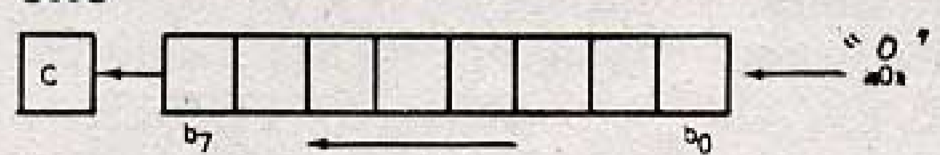
1. Introduction

Nous allons étudier un ensemble d'instructions qui s'apparentent aux instructions logiques puisqu'elles portent leur effet sur les «bits» d'un registre donné. Leurs emplois sont multiples, mais principalement dans les opérations arithmétiques comme la multiplication, la division, l'extraction d'une racine carrée, etc.

Il faut avant d'entreprendre l'étude de cette partie avoir bien compris le mécanisme des registres à décalage.

2. Définitions des «OPERATIONS»

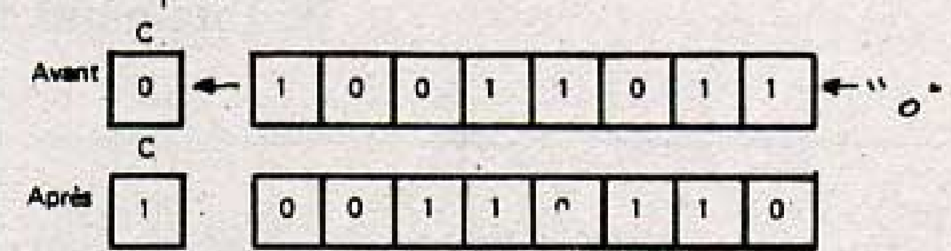
a) Décalage arithmétique à gauche



Description :

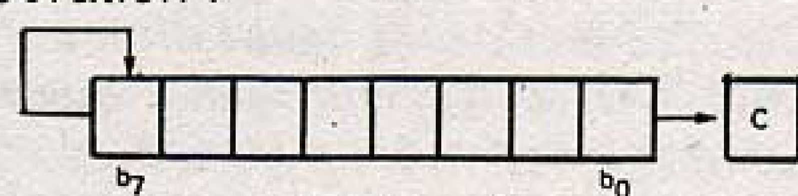
L'exécution de cette instruction décale tous les bits du registre «s» ou de l'octet «m» d'une position **vers la gauche.** L'indicateur C (du registre F) est chargé avec le bit de poids fort du registre «s» ou de l'octet «m». Le bit «b» est chargé avec un zéro.

Exemple :



b) Décalage arithmétique à droite

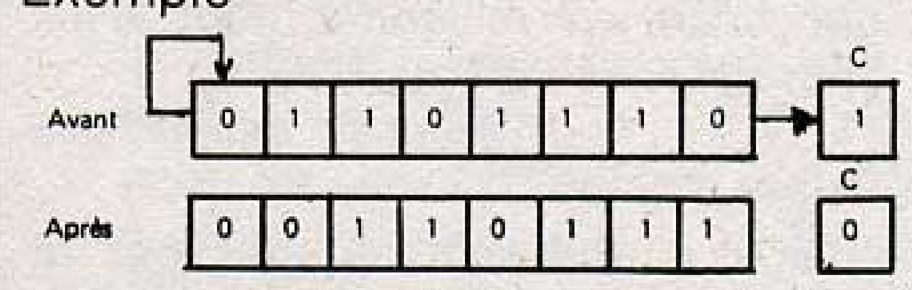
Opération :



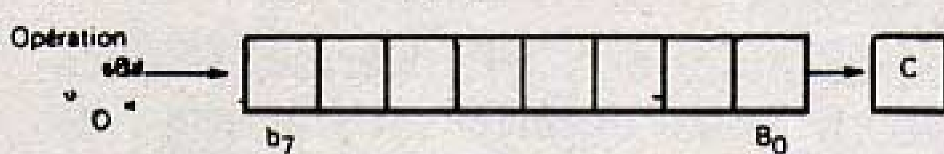
Description :

L'exécution de cette instruction décale tous les bits du registre «s» ou de l'octet «m» d'une position **vers la droite.** Le bit 7 conserve son ancienne valeur. Le bit 0 est chargé dans l'indicateur C (registre F).

Exemple



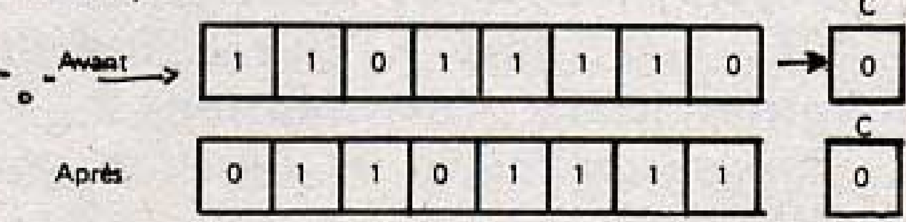
c) Décalage logique à droite



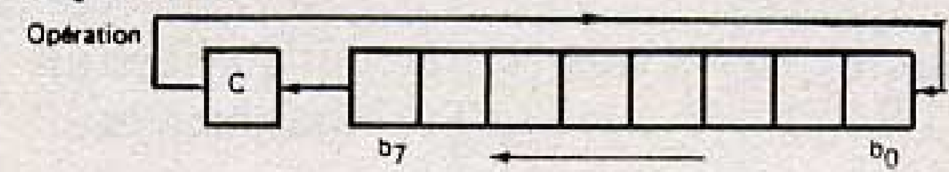
Description :

L'exécution de cette instruction décale tous les bits du registre «s» ou de l'octet «m» d'une position vers la droite. Le bit 7 est chargé avec «0». L'indicateur C (registre F) est chargé avec le bit de poids faible du registre «s» ou de l'octet «m».

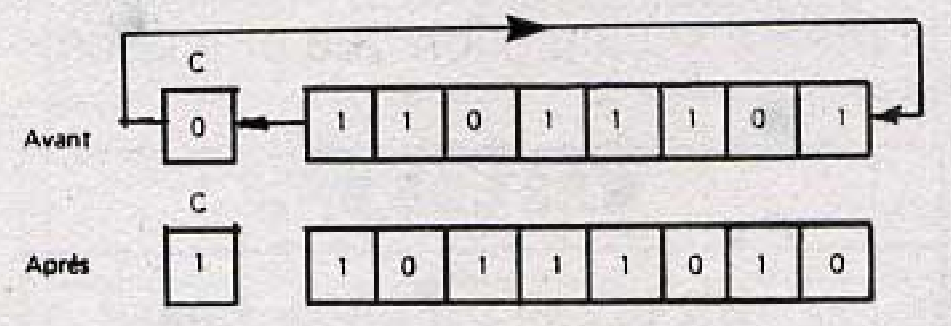
Exemple :



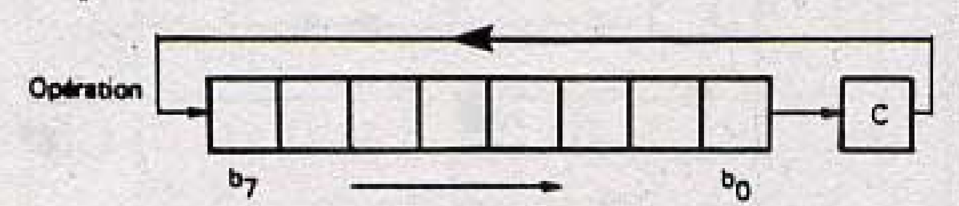
d) Rotation à gauche à travers le report C



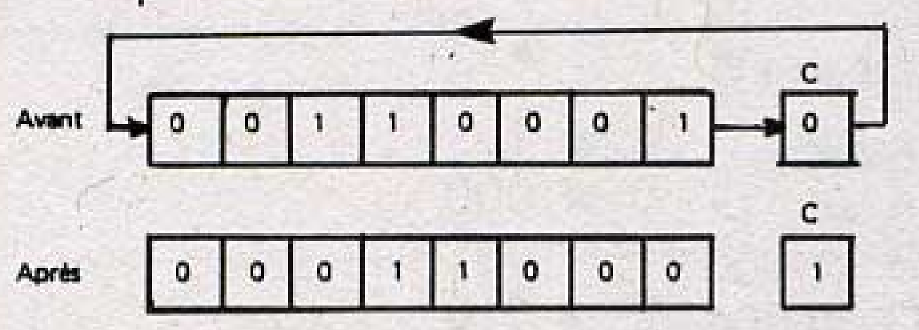
L'exécution de cette instruction décale tous les bits du registre «s» ou de l'octet «m» d'une position vers la gauche. Le bit 0 est chargé avec le bit C. Le bit C est chargé avec le bit de poids fort du registre «s» ou de l'octet «m».



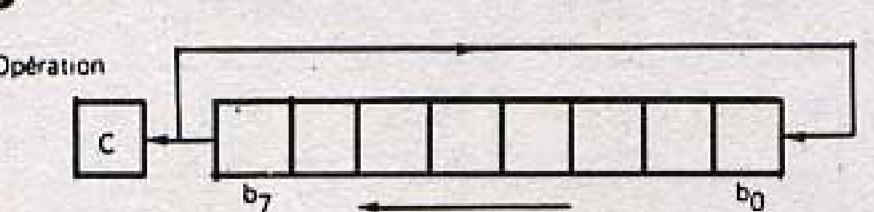
e) Rotation à droite à travers le report



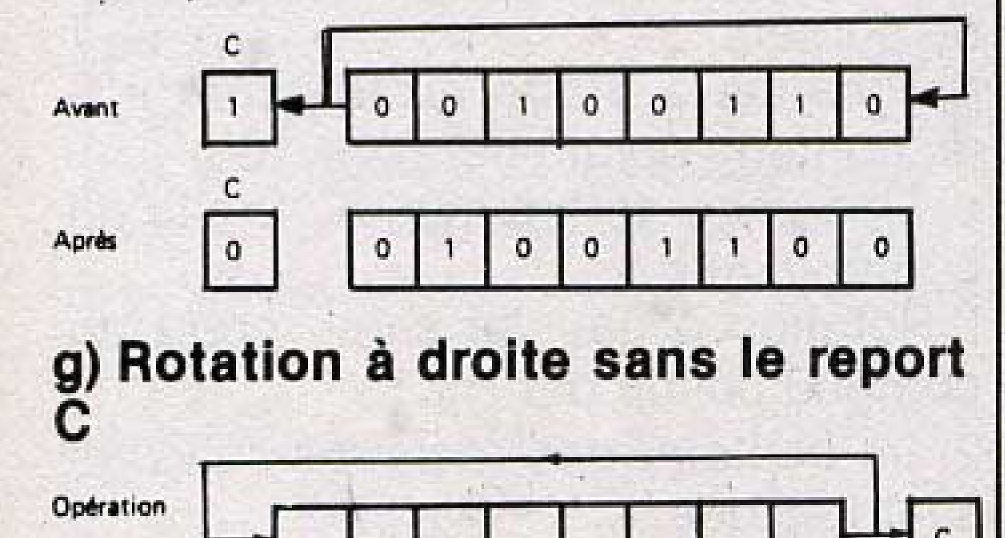
Description :
L'exécution de cette instruction décale tous les bits du registre «s» ou de l'octet «m» d'une position vers la droite. Le bit 7 est chargé avec le bit C. Le bit C est chargé avec le bit de poids faible du registre «s» ou de l'octet «m».
Exemple



f)Rotation à gauche sans le report C

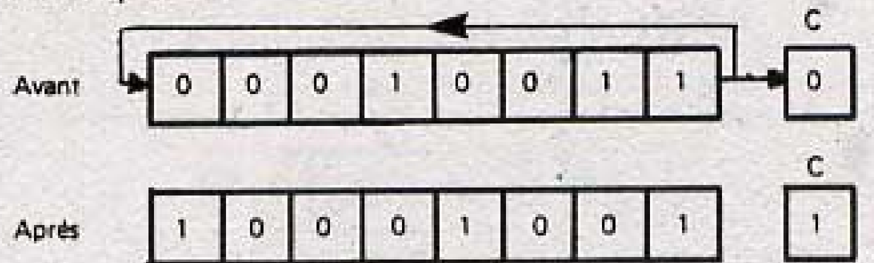


Description :
L'exécution de l'instruction décale circulairement sur lui-même vers la gauche tous les bits du registre «s» ou de l'octet «m». Le bit 7 est chargé dans le bit C ainsi que dans le bit 0 du registre «s» ou de l'octet «m».
Exemple :



g) Rotation à droite sans le report C

Description :
L'exécution de l'instruction décale circulairement sur lui-même vers la droite tous les bits du registre «s» ou de l'octet «m». Le bit b_0 est chargé dans le bit C ainsi que dans le bit 7 du registre «s» ou de l'octet «m».
Exemple :



h) Codes mnémoniques
Dans la définition des opérations de décalage et de rotation, nous n'avons pas précisé ce qu'était le registre «s» ou l'octet «m».
Comme pour les opérations logiques et arithmétiques, l'opérande :
«s» désigne l'un des registres A, B, C, D, E, H ou L.
«m» désigne un octet (une case mémoire) pointée par le contenu de la paire de registres HL ou par le contenu de l'un des registres index IX ou IY auquel s'ajoute le déplacement d.
L'ensemble du répertoire des instructions de décalage et de rotation, se trouve ainsi résumé dans les quatre tableaux suivants (I à IV).

III. MANIPULATION ET TEST DE BITS

1. Exemple
Supposons que le contenu d'une case mémoire doit être pour la suite du déroulement du programme une quantité paire (donc le bit de poids faible à «0»). Pour cela il suffit d'effectuer un ET logique avec le contenu de cette case et le contenu de A précédemment chargé avec FE (1111 1110), puis de replacer le tout dans la case d'origine.
Il peut être aussi nécessaire de tester un bit particulier d'un registre ou d'une case mémoire. Et pour ce faire, il suffit d'employer les instructions logiques adéquates.
En réalité, dans le Z 80, des instructions dites «manipulation de bits» et «test de bits» existent «toutes faites» et permettent ainsi de «soulager» d'autant les programmes.

2. Manipulations de bits
L'instruction RES b_i, s ou RES b_i, m permet de mettre le bit désigné ($b_0 \leqslant b_i \leqslant b_7$) à l'état «0» ou effacement. Dans cette instruction l'opérande :
«s» désigne l'un des registres A, B, C, D, E, H ou L
«m» désigne un octet (une case mémoire) pointée par le contenu de la paire de registres HL ou par le contenu de l'un des registres index IX ou IY auquel s'ajoute le déplacement d.
L'instruction SET b_i, s ou SET b_i, m permet de mettre le bit désigné (b_0 b_i b_7) à l'état «1». Dans cette instruction l'opérande a la même signification que précédemment.
L'ensemble du répertoire des instructions de manipulations de bits, se trouve résumé dans le tableau V.

DECALAGE ARITHMETIQUE					
SL A, «s» ou SL A, «m»			SR A, «s» ou SR A, «m»		
C ← b_7 ← b_0 ← «0»			→ b_7 → b_0 → C		
r	1 1 0 0 1 0 1 1 0 0 1 0 0 r	CB 2–	r	1 1 0 0 1 0 1 1 0 0 1 0 1 r	CB 2–
(HL)	1 1 0 0 1 0 1 1 0 0 1 0 0 1 1 0	CB 26	(HL)	1 1 0 0 1 0 1 1 0 0 1 0 1 1 1 0	CB 2E
(IX + d)	1 1 0 1 1 1 0 1 1 1 0 0 1 0 1 1 ← d → 0 0 1 0 0 1 1 0	DD CB 26	(IX + d)	1 1 0 1 1 1 0 1 1 1 0 0 1 0 1 1 ← d → 0 0 1 0 1 1 1 0	DD CB 2E
(IY + d)	1 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 ← d → 0 0 1 0 0 1 1 0	FD CB 26	(IY + d)	1 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 ← d → 0 0 1 0 1 1 1 0	FD CB 2E

avec r : A = 111 D = 010 L = 101
B = 000 E = 011
C = 001 H = 100

DECALAGE LOGIQUE			
SRL «s» ou SRL «m»			
«0» → b_7 b_0 → C			
r	1 1 0 0 0 0 1 1	1 0 1 1 1 r	CB 3–
(HL)	1 1 0 0 0 0 1 1	1 0 1 1 1 1 1 0	CB 3E
(IX + d)	1 1 0 1 1 1 0 0 d 0 0 1 1	1 1 0 1 1 0 1 1 1 1 1 0	DD CB 3E
(IY + d)	1 1 1 1 1 1 0 0 d 0 0 1 1	1 1 0 1 1 0 1 1 1 1 1 0	FD CB 3E

Tableau II

Avec r : A = 111 D = 010 L = 101
B = 000 E = 011
C = 001 H = 100

ROTATION A TRAVERS LE REPORT C			
RL «s» ou RL «m»			
C ← b_7 ← b_0 ← (retour vers C)			
r	1 1 0 0 0 0 0 1 0	1 0 1 1 r	CB 1–
(HL)	1 1 0 0 0 0 0 1	1 0 1 1 0 1 1 0	CB 16
(IX + d)	1 1 0 1 1 1 0 0 ← d → 0 0 0 1	1 1 0 1 1 0 1 1 0 1 1 0	DD CB 16
(IY + d)	1 1 1 1 1 1 0 0 ← d → 0 0 0 1	1 1 0 1 1 0 1 1 0 1 1 0	FD CB 16

RR «s» ou RR «m»			
→ b_7 → b_0 → C (retour vers b_7)			
r	1 1 0 0 0 0 0 1	1 0 1 1 1 r	CB 1–
(HL)	1 1 0 0 0 0 0 1	1 0 1 1 1 1 1 0	CB 1E
(IX + d)	1 1 0 1 1 1 0 0 ← d → 0 0 0 1	1 1 0 1 1 0 1 1 1 1 1 0	DD CB 1E
(IY + d)	1 1 1 1 1 1 0 0 ← d → 0 0 0 1	1 1 0 1 1 0 1 1 1 1 1 0	FD CB 1E

Tableau III

avec r : A = 111 D = 010 L = 101
B = 000 E = 011
C = 001 H = 100

3. Test de bit

L'instruction Bit b_i, s ou Bit b_i, m teste le bit désigné ($b_0 \leqslant b_i \leqslant b_7$). L'indicateur Z du registre F contiendra le complément (ou l'INVERSE) du bit ainsi désigné.

Dans cette instruction, l'opérande a la même signification que précédemment.

TEST DE BIT			
Bit b_i «s» ou bit b_i «m»			
r	1 1 0 0 0 1 ←	1 0 1 1 b → ← r →	CB
(HL)	1 1 0 0 0 1 ←	1 0 1 1 b → 1 1 0	CB
(IX + d)	1 1 0 1 1 1 0 0 0 1 ← ←	1 1 0 1 1 0 1 1 b → 1 1 0 d →	DD CB
(IY + d)	1 1 1 1 1 1 0 0 ← 0 1 ←	1 1 0 1 1 0 1 1 d → b → 1 1 0	FD CB

Tableau VI

Avec r :	avec b :
A = 111	0 = 000
B = 000	1 = 001
C = 001	2 = 010
D = 010	3 = 011
E = 011	4 = 100
H = 100	5 = 101
L = 101	6 = 110
	7 = 111

4. Exercices

1. Ecrire les instructions permettant de réaliser les opérations suivantes :

a) Rotation à gauche sans report du contenu de D

b) Décalage logique du contenu pointé par HL

c) Décalage arithmétique à gauche du contenu de A

d) Rotation à droite au travers de C du contenu pointé par (IX + 10d)

e) Décalage arithmétique à droite du contenu de E

f) Rotation à gauche sans report du contenu pointé par HL.

2. Le contenu des registres suivants sont tels que :

A = 8B D = EC
B = C1 E = DD
C = 9C F = 19

HL = 1A 3E et le contenu de la case d'adresse mémoire 1A3E : 5A.

Quels sont les contenus des registres et le contenu de l'adresse 1A3E,

ROTATION SANS LE REPORT C					
RLC «s» ou RLC «m»			RRC «s» ou RRC «m»		
C ← b7 ← b0			→ b7 → b0 → C		
r	1 1 0 0 1 0 1 1 0 0 0 0 0 r	CB 0–	r	1 1 0 0 1 0 1 1 0 0 0 0 1 r	CB 0–
(HL)	1 1 0 0 1 0 1 1 0 0 0 0 0 1 1 0	CB 06	(HL)	1 1 0 0 1 0 1 1 0 0 0 0 1 1 1 0	CB 0E
(IX + d)	1 1 0 1 1 1 0 1 1 1 0 0 1 0 1 1 ← d → 0 0 0 0 0 1 1 0	DD CB 06	(IX + d)	1 1 0 1 1 1 0 1 1 1 0 0 1 0 1 1 ← d → 0 0 0 0 1 1 1 0	DD CB 0E
(IY + d)	1 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 ← d → 0 0 0 0 0 1 1 0	FD CB 06	(IY + d)	1 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 ← d → 0 0 0 0 1 1 1 0	FD CB 0E

Tableau IV

Avec r : A = 111 D = 010 L = 101
B = 000 E = 011
C = 001 H = 100

MANIPULATION DE BIT					
Res b_i, s ou Res b_i, m			Set b_i, s ou Set b_i, m		
r	1 1 0 0 1 0 1 1 1 0 ← b → ← r →	CB	r	1 1 0 0 1 0 1 1 1 1 ← b → ← r →	CB
(HL)	1 1 0 0 1 0 1 1 1 0 ← b → 1 1 0	CB	(HL)	1 1 0 1 0 1 1 1 1 ← b → 1 1 0	CB
(IX + d)	1 1 0 1 1 1 0 1 1 1 0 0 1 0 1 1 ← d → 1 0 ← b → 1 1 0	DD CB	(IX + d)	1 1 0 1 1 1 0 1 1 1 0 0 1 0 1 1 ← d → 1 1 ← b → 1 1 0	DD CB
(IY + d)	1 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 ← d → 1 0 ← b → 1 1 0	FD CB	(IY + d)	1 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 ← d → 1 1 ← b → 1 1 0	FD CB

Tableau V

avec r : A = 111 E = 011
B = 000 H = 100
C = 001 L = 101
D = 010

b : 0 – 000 4 – 100
1 – 001 5 – 101
2 – 010 6 – 110
3 – 011 7 – 111

après l'exécution des instructions suivantes :
CB 1A
CB 07
CB 2E
CB 38
CB 14
Notez le contenu de C après chaque instruction.

3. Indiquez les codes opératoires qui réalisent les instructions suivantes :
a) Mettre à 1 le bit 3 de E
b) Mettre à «0» le bit 7 de A
c) Mettre à «0» le bit 2 de F
d) Tester le bit 5 de C
e) Mettre à «0» le bit 7 de l'emplacement pointé par (IX + 09)
f) Mettre à «1» le bit 4 de B
g) Mettre à «1» le bit 5 de D
h) Tester le bit 0 de H

IV. INSTRUCTIONS «CONTROLE DU CPU»

Les instructions qui vont suivre agissent directement sur l'unité centrale et contrôlent son fonctionnement.

1. Instruction NOP

La première est l'instruction NOP qui signifie «pas d'opération» (NO OPERATION) dont le code hexadécimal est 00.

Pendant l'exécution de cette instruction, seul **le déroulement du cycle du microprocesseur** a lieu. Le microprocesseur se trouve dans une position de «Stand-by». Il génère les signaux normalement, notamment ceux indispensables au rafraîchissement des mémoires dynamiques RAM. Le compteur ordinal est incrémenté. **Aucune «opération» n'est effectuée.**

Une telle instruction, qui apparemment «ne fait rien» est sans objet. Eh bien non ! Nous allons donner deux exemples qui montrent l'intérêt de cette instruction.

Cas 1

Au cours de la mise au point d'un programme, il arrive bien souvent qu'il soit nécessaire de supprimer un ou plusieurs octets, sans pour autant réécrire tout le programme. Il peut s'agir d'instructions en trop ou d'instructions que l'on veut momentanément supprimer pour effectuer des essais de «mise au point».

Dans ce cas, il suffit de remplacer les octets concernés par des instructions «NOP» sans autre changement.

C'est une manière de «gommer» provisoirement ou définitivement une ou plusieurs instructions.

Cas 2

Une deuxième application dans laquelle on trouve fréquemment les instructions NOP sont les boucles de temporisation.

Le registre B ayant été chargé avec N (figure 146), par exemple 64H (soit 100d), le programme sera temporairement suspendu pendant l'exécution de 2 × N fois l'instruction «NOP».

Si l'horloge est de 4 MHz, le temps d'exécution de NOP est de 1 μs, ce qui va donner lieu à une temporisation d'environ 525 μs (le T.E. de DJNZ étant de 3,25 μs avec l'horloge à 4 MHz).

Pour augmenter ou diminuer le «temps de suspension» du déroulement du programme, il suffit d'agir sur le nombre N ou la quantité de NOP introduite dans la boucle.

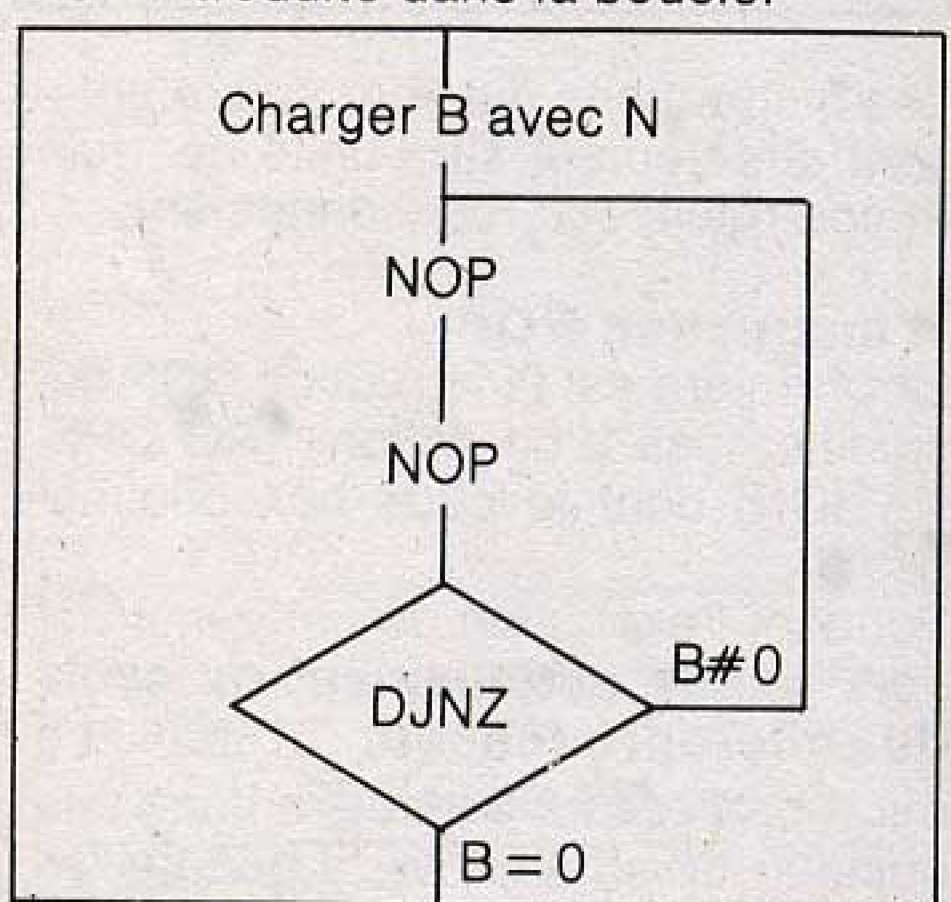


Figure 146

2. Instruction «HALT»

Une deuxième méthode pour suspendre le déroulement d'un programme est l'instruction HALT dont le code hexadécimal est 76.

L'exécution de l'instruction HALT amène le CPU à **exécuter continuellement les instructions NOP :** c'est une **boucle de temporisation sans fin.** Le CPU ne peut sortir de lui-même de cette boucle ; il faut **un stimuli provenant d'une commande extérieure.**

En reprenant la configuration hardware, telle qu'elle fut présentée figure 44 (LM 11, p. 49), le nombre d'entrées qui peuvent fournir l'impulsion qui permettra au CPU **de continuer est en réalité** limité à trois. Ce ne peut être que : Reset, NMI ou INT.

Une commande RESET (active sur le flanc descendant) force le compteur ordinal à 00 H. Le CPU démarre comme s'il s'agissait d'une première initialisation.

Une commande NMI (active sur le flanc descendant) force le compteur ordinal à se rendre en 0066 H (saut en première page). Le contenu du compteur ordinal (qui pointe l'instruction qui suit HALT) est sauvegardé dans la pile.

Après exécution de la routine, placée en 0066 H, le déroulement du programme reprend là où il avait été interrompu par l'exécution de l'instruction HALT.

Une commande INT (active sur le flanc descendant) permet la poursuite du fonctionnement du CPU, si et seulement si les interruptions ont été préalablement autorisées par le logiciel (donc inclus dans le programme).

Ce signal INT est souvent généré par le ou les circuits d'entrée/sortie comme nous le montrerons dans l'exemple ci-après.

Lorsque le CPU honore une demande d'interruption, il peut y répondre selon l'un des trois modes qui seront décrits en détails dans le paragraphe suivant l'exemple.

Exemple :

Une imprimante de vingt caractères de type thermique par exemple, est connectée à un système par l'intermédiaire d'un circuit périphérique d'entrée/sortie (figure 147).

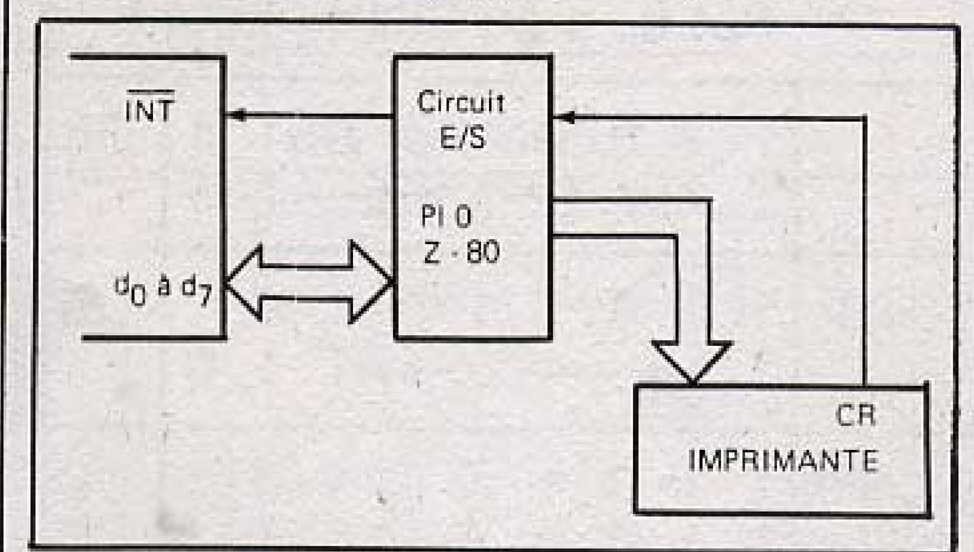


Figure 147

La vitesse de l'imprimante est de 4 lignes/seconde, ce qui correspond à un temps d'impression de 250 ms par ligne.

L'imprimante possède un registre d'entrée (ou buffer) qui stocke la ligne à imprimer.

Le temps de chargement du registre BUFFER qui est en fait un transfert de 20 octets est de l'ordre de 3 à 400 μs.

Dans le programme, après l'envoi des 20 caractères qui représentent une ligne, une instruction HALT sera placée : le CPU attendra le message «fin de ligne» ou CR (Carriage Return ou retour chariot) pour envoyer le bloc de caractères suivant. Et ainsi de suite jusqu'à ce que tout le texte soit transmis.

Pourquoi ne pas remplacer l'instruction HALT par une boucle de temporisation ?

Le temps d'impression de 250 millisecondes correspond à un régime continu. Pour un fonctionnement ligne par ligne, le temps est plus long. D'autre part, il s'agit là d'une valeur typique, la valeur réelle variant d'une imprimante à l'autre, ainsi que tout au long de sa durée de vie. Si bien que si on utilisait une bouche de temporisation, il faudrait utiliser le temps maximal, et par conséquent au détriment des performances de rapidité de l'imprimante.

Par contre, en utilisant l'instruction HALT, **notre système s'adapte automatiquement** au temps d'impression de l'imprimante ; ce qui permet d'obtenir dans tous les cas la vitesse optimale, quelle que soit l'imprimante, son mode de fonctionnement ou son évolution dans le temps.

3. Instructions d'autorisation et d'interdiction des interruptions

Pour qu'une demande d'interruption masquable soit honorée par le CPU, il faut au préalable que le logiciel l'autorise. Cette autorisation s'effectue par l'exécution de l'instruction EI (Enable Interrupt) dont le code hexadécimal est : FB.

Inversement, le programmeur peut à tout moment interdire l'autorisation des interruptions (par exemple pendant une boucle de temporisation). Pour cela, il place dans son programme l'instruction DI (Disable Interrupt) dont le code hexadécimal est : F3.

A noter qu'à la mise sous tension (et après une action RESET) les interruptions ne sont pas autorisées.

4. Modes d'interruptions

Le Z80 possède trois modes différents d'interruption qui peuvent être sélectionnés par 3 instructions :

IM0 ou Mode 0 (ED 46)

Dans ce mode d'exploitation, le circuit périphérique générant l'interruption place en même temps un octet sur le bus de donnée correspondant à l'un des codes des instructions Restart.

Le CPU autorise le compteur ordinal à se charger avec cette adresse après avoir effectué une sauvegarde dans la pile.

IM1 ou Mode 1 (ED 56)
Ce mode est une version simplifiée du précédent. Le CPU exécute automatiquement un saut à l'adresse 0038 H (Rest 56), en ayant au préalable sauvegardé le contenu du PC dans la pile.

IM2 ou Mode 2 (ED 5E)
Ce mode d'exploitation est, par contre, plus sophisitiqué et beaucoup plus puissant que les deux autres. **Il permet au CPU de se rendre à n'importe quelle adresse paire du champ d'adresse.**
Initialement, le registre I (le registre Interruptions) a été chargé avec les 8 bits de poids fort de l'adresse à atteindre. Tandis que les 8 bits de poids faible (en réalité 7, le plus faible étant 0) **sont fournis pas le circuit interrompant.**
Le CPU forme ainsi une adresse de 16 bits qui sera chargée dans le compteur ordinal après que son contenu soit sauvegardé dans la pile.
Le tableau suivant (figure 148) résume les différentes instructions qui permettent le contrôle du CPU.

NOP	00
HALT	76
EI	F3
DI	FB
IMO	ED-46
IMI	ED-56
IM2	ED-5E

Figure 148

V. INSTRUCTIONS ENTREE-SORTIE

1. Introduction
Tous les échanges que nous avons étudiés jusqu'à présent s'effectuaient soit à l'intérieur même du microprocesseur, échanges entre registres, soit avec les mémoires ROM's ou RAM's.

Pour ces derniers, le canal utilisé est le «Bus de données». Dans cette partie nous allons étudier les échanges avec les unités extérieures.

Le transfert des informations qu'elles soient des données d'«entrée» ou de «sortie» transitent toujours par le même canal «le bus de données». Il est inconcevable qu'une unité périphérique (surtout si elle est lente) immobilise cette ligne de communication pendant tout le temps nécessaire pour accomplir sa tâche. C'est pourquoi un circuit d'interface est généralement placé entre le système et le périphérique de manière à ne pas nuire à l'efficacité du CPU. Le tout restant bien entendu **sous le contrôle du CPU par des échanges de signaux via le bus de commandes.**
Un boîtier d'interface (40 broches) peut contenir 2 (PIO – Z80) ou 3 (8255, Intel) circuits d'entrées-sorties parallèles (figure 12, LM 9).
La configuration fonctionnelle d'un circuit d'interface est programmée lors de l'initialisation du système. Chaque circuit doit être programmé en «entrée» ou en «sortie» ou bien encore en «entrée-sortie» avec les lignes de commandes correspondantes.

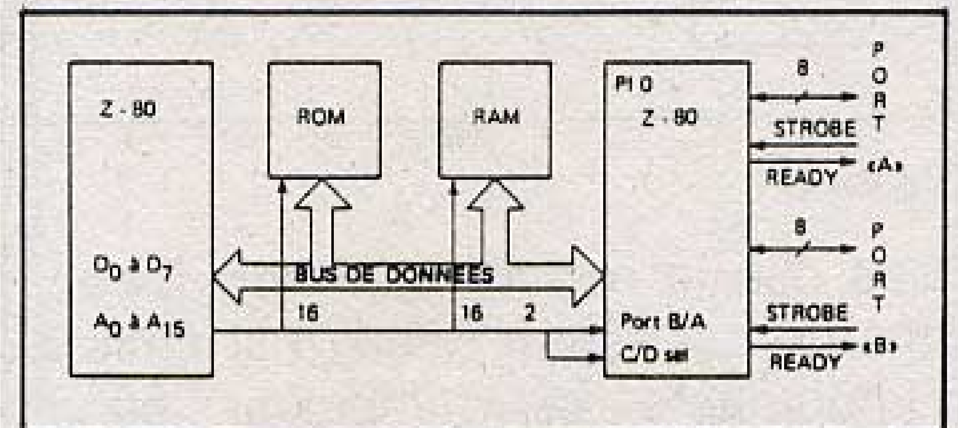


Figure 149

Pour une bonne compréhension des instructions qui vont suivre, nous décrirons succinctement le circuit PIO-Z80. Ce circuit d'interface est essentiellement constitué de deux circuits (deux voies) indépendants appelés «PORT A» et «PORT B».
Chaque «PORT» est avant tout un registre 8 bits, bidirectionnel. Il peut être mis en liaison avec le «bus de données» donc avec le CPU au moyen de signaux d'échange générés par le Z80 (donc sans contrôle du programme).
La sélection du port s'effectue au moyen d'un fil d'adresse, noté «Port B/A sel». Lorsque le niveau est bas, le port B est sélectionné ; lorsque le niveau est haut, le port A est sélectionné.
La configuration fonctionnelle que l'on veut donner à chaque port s'effectue **à l'aide du mot de contrôle,** placé dans le registre de commande du circuit d'interface. Une deuxième entrée, notée : «C/D sel» permet de charger ce registre.
Un niveau bas sur cette broche signifie que le bus de données est utilisé **pour transférer une donnée entre le CPU et l'un des «ports»,** tandis qu'un niveau haut signifie que **la donnée est un mot de commande.**

Cette fonction de contrôle est souvent assurée par le bit A_1 du bus d'adresse, tandis que le choix du port est souvent assuré par le bit A_0 (figure 150).

	A_1	A_0	«n»
Port A	0	1	1
Port B	0	0	0
Mot de commande	1	X	

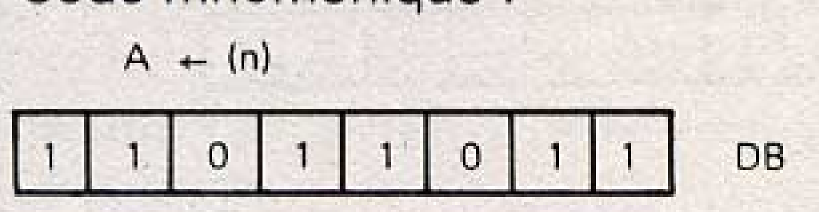
Figure 150

Le mot de commande définit entre autres, si un port donné travaille en mode :
- ENTREE
- SORTIE
- BIDIRECTIONNEL

Deux signaux d'échanges (READY et STROBE) supplémentaires sont disponibles au niveau de chaque port pour synchroniser les échanges avec l'extérieur.
Vis-à-vis du CPU, les «ports» se comportent comme l'une quelconque des cases mémoires (ainsi que le registre de commande). Cependant, comme les signaux de contrôle sont spécifiques par rapport à un emplacement mémoire, les instructions seront :
OUT (ou sortie) pour l'écriture d'un mot
IN (ou entrée) pour la lecture d'une donnée.
Généralement, un microprocesseur peut sélectionner jusqu'à 256 registres (entrée-sortie ou commande), ce qui revient à dire que seuls les 8 fils du bus d'adresse (A_0 à A_7) sont utilisables.
L'adresse du registre (figure 150) sera noté «n» avec $0 \leqslant «n» \leqslant 255$.

2. Instructions d'ENTREE
a) Chargement de l'accumulateur à partir d'un port (n)
Opération :
Le contenu du port (n) est chargé dans l'accumulateur A.
Le contenu de A apparaît sur les bits d'adresses A_8 à A_{15}.

Code mnémonique :



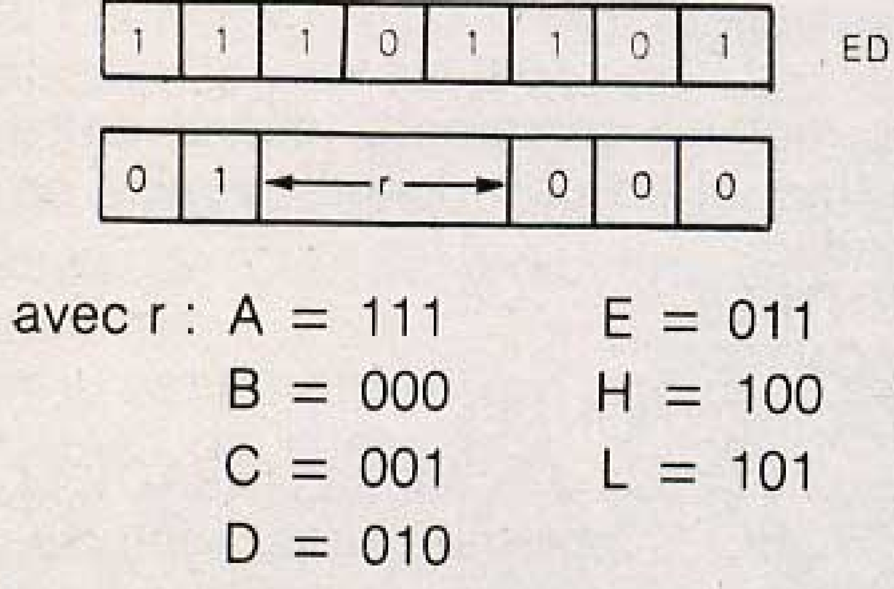
Code hexadécimal :



b) Chargement d'un registre r à partir d'un port désigné par le contenu de C
Opération :
Le contenu du port (n) est chargé dans l'un des registres A, B, C, D, E, H ou L. L'adresse (n) du port est l'octet contenu dans le registre C.

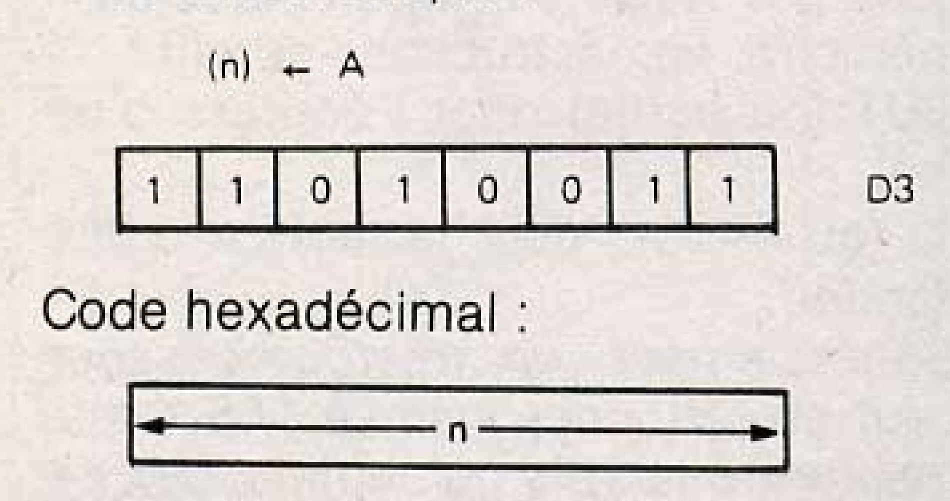
Code mnémonique : r ← (C)

Codes machines :



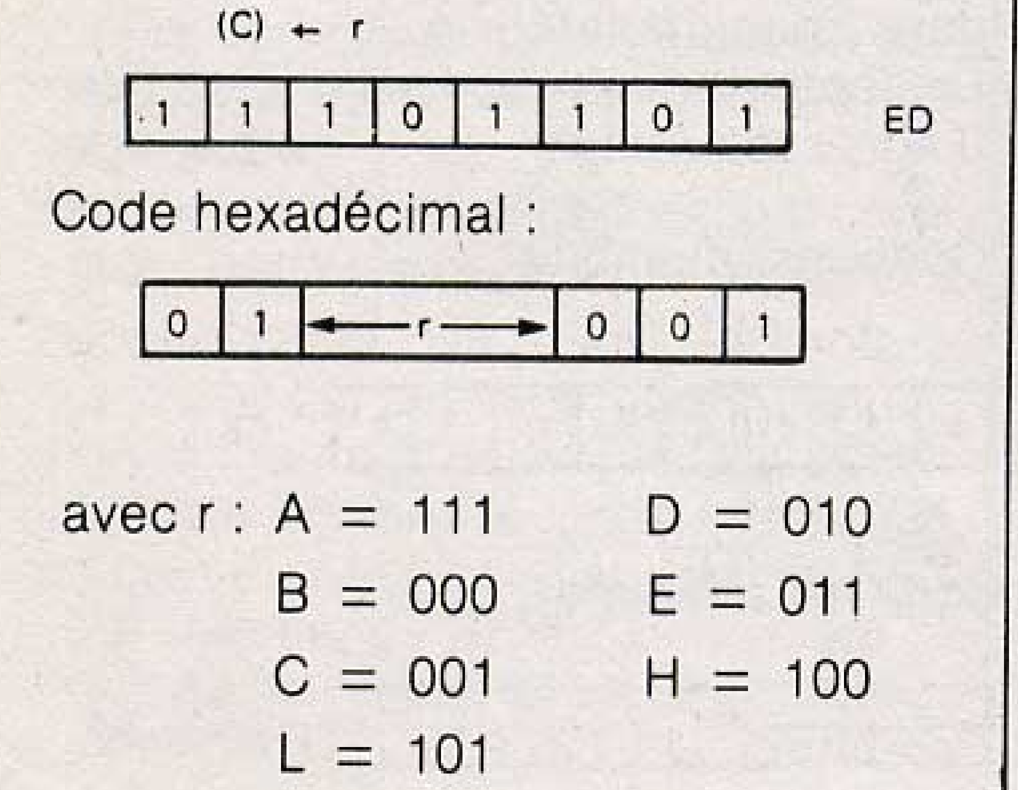
3. Instructions de SORTIE
a) Transfert du contenu de A dans un port (n)
Opération :
Le contenu de l'accumulateur A est transféré dans le port «n».

Code mnémonique :



b) Transfert du contenu de r dans un port désigné par le contenu de C
Opération :
Le contenu du registre r est transféré dans le port d'adresse «n». L'adresse n étant l'octet contenu dans le registre C.

Code mnémonique :



4. Exemples d'application
a) Exemple 1
Problème :
Une imprimante de 40 caractères est connectée au port B d'un circuit d'interface PIO-Z80. Chaque caractère est représenté en code ASCII par 1 octet. La ligne à imprimer est stockée à partir de l'adresse 1800 H à 1827 H (40 emplacements) (figure 151).
Ecrire le programme qui réalise le transfert.
1.a. Aspect hardware

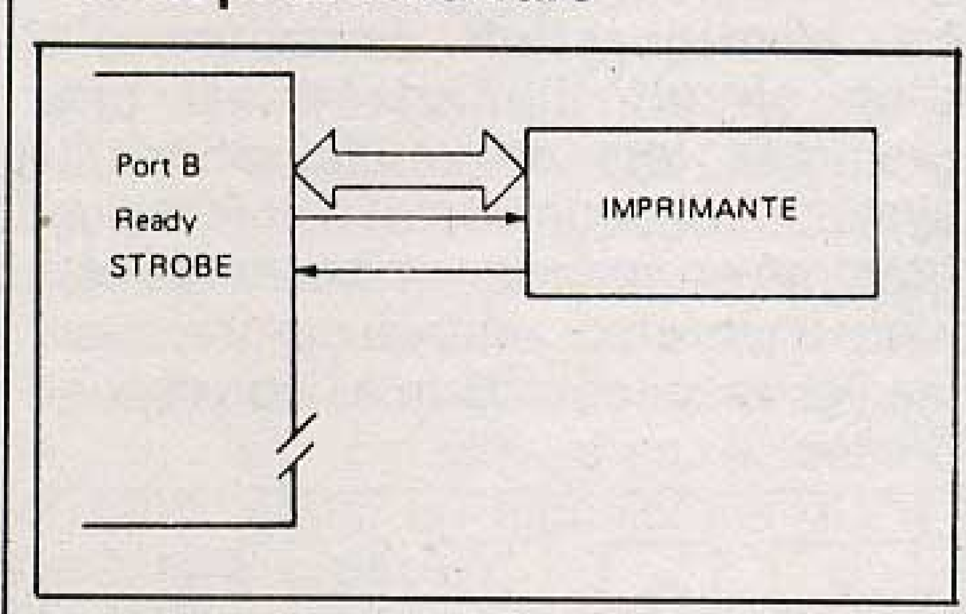


Figure 151

Dès que le registre du «Port B» est chargé, un signal Ready est émis à l'imprimante pour indiquer qu'un octet est prêt. Quand celle-ci après avoir prélevé le mot, l'aura placé dans le registre, elle envoie un signal «STROBE» pour signifier qu'elle est prête à accepter le caractère suivant.
Ainsi, les 40 caractères sont transférés dans le registre imprimante, la ligne est alors imprimée, l'imprimante est prête pour la suivante.
1.b. Aspect software
Le programme est présenté en mnémonique par la figure 152.
Les 3 premières instructions constituent la partie «initialisation».
Quand le transfert de la case mémoire pointée par HL dans l'accumulateur a eu lieu, la paire de registres HL est incrémentée. Le contenu de A, c'est-à-dire 1 caractère, est transféré dans le port de sortie.
Etant donné que le temp nécessaire à l'imprimante pour stocker et ranger l'octet est de quelques dizaines de microsecondes (40 environ), il va falloir suspendre le déroulement du programme par une instruction HALT Cependant, il faut prendre soin auparavant de **valider les interruptions.**
Lorsque l'imprimante renvoie le message «STROBE» (message d'acquittement), une «demande d'interruption» est émise par le circuit d'interface. Les interruptions étant validées, le

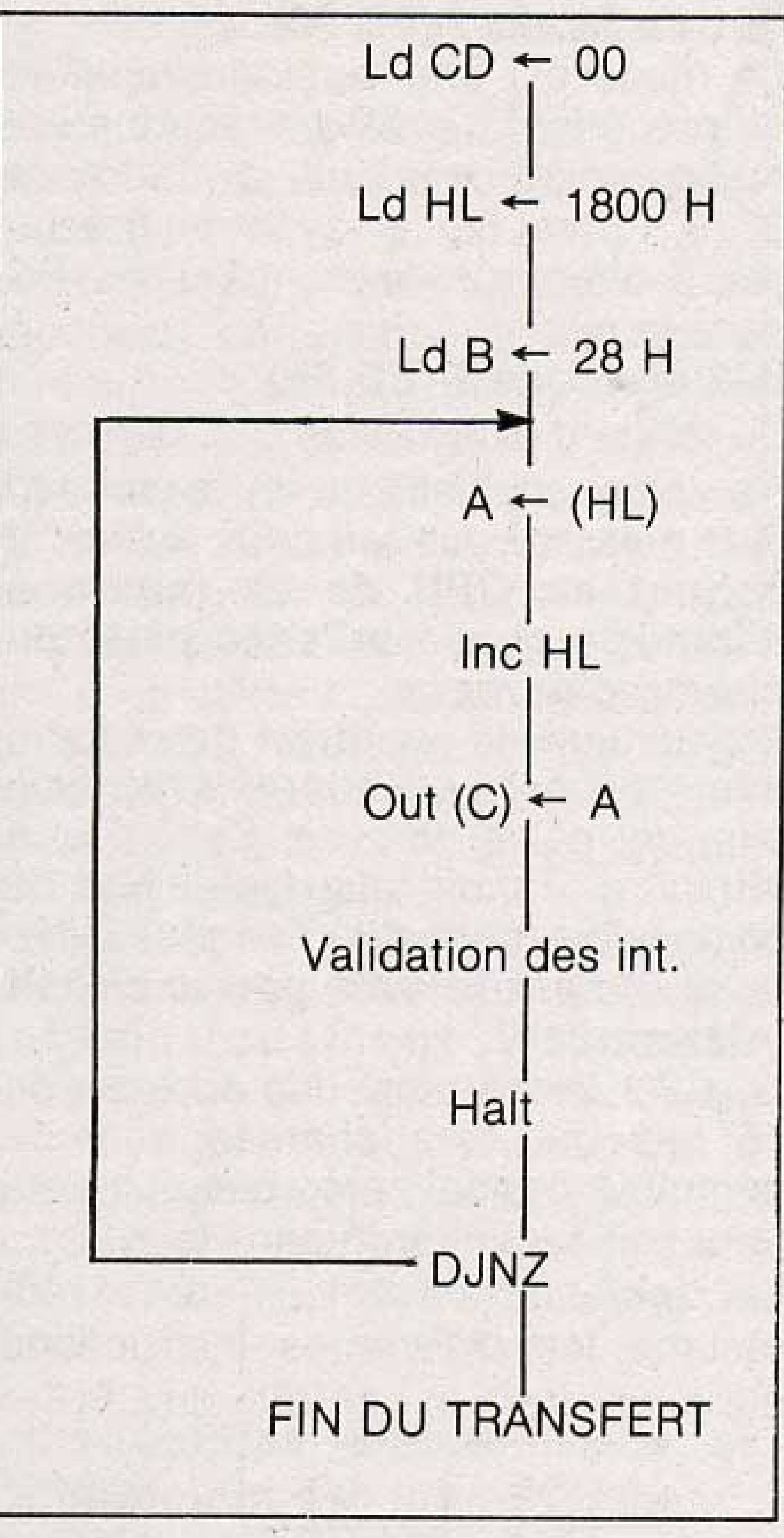


Figure 152

CPU peut l'honorer. Dans le cas présent, il suffit de poursuivre le programme (exécution de DJNZ).
Donc nous choisissons le mode d'interruption le plus simple, le mode 1, qui charge le compteur ordinal avec 0038 H. Dans cette case mémoire nous aurons placé au préalable l'instruction RET (C9).
Le PC sera chargé avec les 2 octets de la pile, c'est-à-dire l'adresse de l'instruction DJNZ.
Pour une parfaite compréhension de la boucle, nous aurions pu écrire celle-ci sous la forme de la figure 153.
Avant d'étudier les instructions de «transfert par bloc» qui peuvent simplifier notre problème, étudions un second exemple dérivé du précédent.

2. Exemple 2
Problème
Le système microprocesseur est le cœur d'un banc de test, qui reçoit des paramètres, les traite et périodiquement les envoie ensuite par bloc de 100 octets à un mini-ordinateur connecté à ce banc de test (figure 154). (Par exemple, le mini-ordinateur peut être connecté à 10

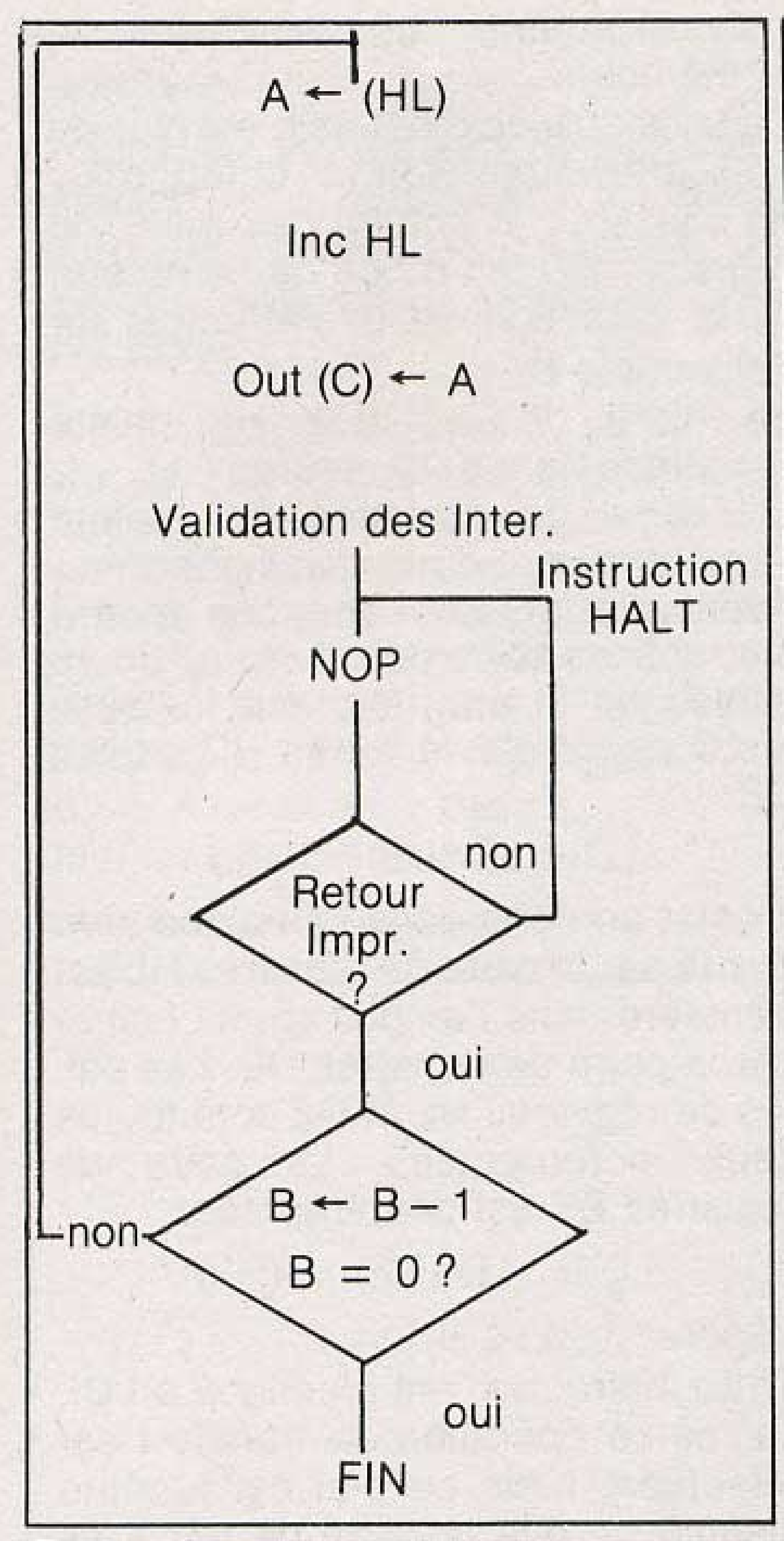


Figure 153

bancs de test : il centralise les informations déjà traitées par le système à microprocesseur).
Le temps de saisie d'un octet par le mini-ordinateur étant de 1 microseconde, **il n'est pas nécessaire d'inclure dans le programme une boucle de temporisation (Halt).**
Ecrire le programme correspondant pour ce problème.

2.a. Aspect hardware

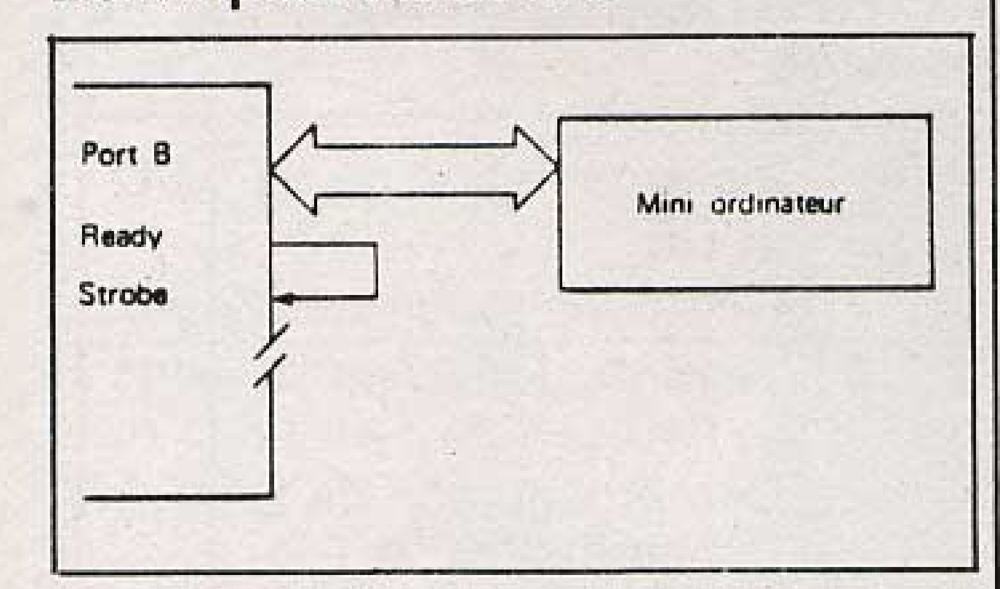


Figure 154

Dès que le registre du «Port B» est chargé, l'octet est saisi par le mini-ordinateur sans qu'il soit nécessaire d'attendre. (En réalité, le temps nécessaire pour placer un autre octet de l'ordre de 10 µs est largement suffisant au temps de saisie par le mini 1 µs).

On notera que les signaux Ready (sortie) et Strobe (entrée) sont connectés entre eux.

2.b. Aspect software

Le programme est présenté en mnémoniques par la figure 155. A noter que pendant la boucle de transfert, les interruptions sont interdites (DI). Lorsque la boucle est terminée, c'est-à-dire que le bloc de 100 octets (64 H) a été transféré, les interruptions sont à nouveau autorisées.

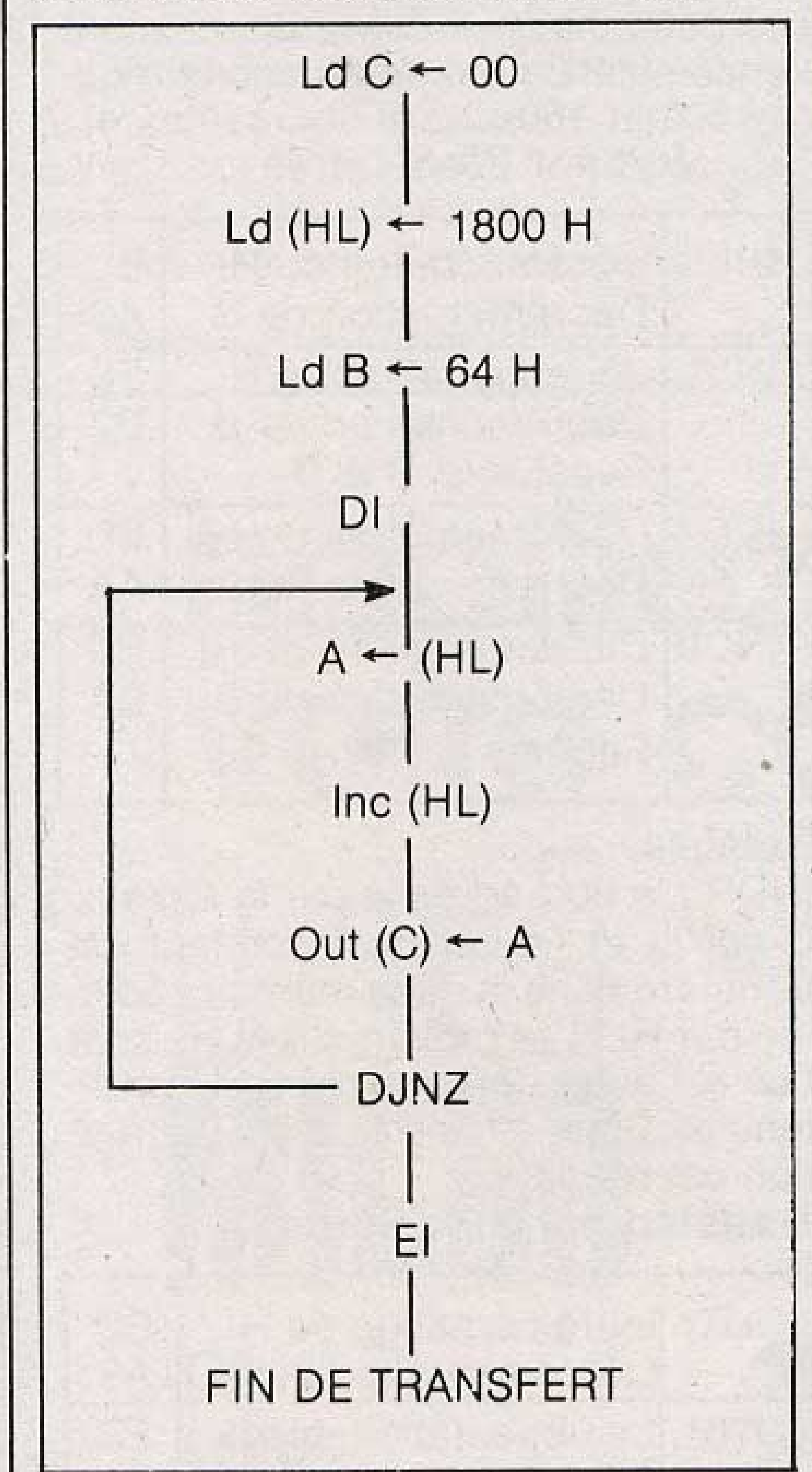


Figure 155

5. Instructions de transfert par bloc

L'ensemble des 4 instructions qui constituent la boucle de transfert peut être remplacé par une instruction unique notée OTIR dont le code hexadécimal est ED 83.

Description

Le contenu de l'emplacement mémoire pointé par la paire de registres HL est transféré dans le Port de sortie adressé par le contenu du registre C. Le registre B est décrémenté, et le registre HL incrémenté.

Si B # 0, l'instruction est répétée (compteur ordinal décrémenté de 2).

Si B = 0, l'instruction suivante est exécutée.

Le programme est ainsi réduit à celui de la figure 156.

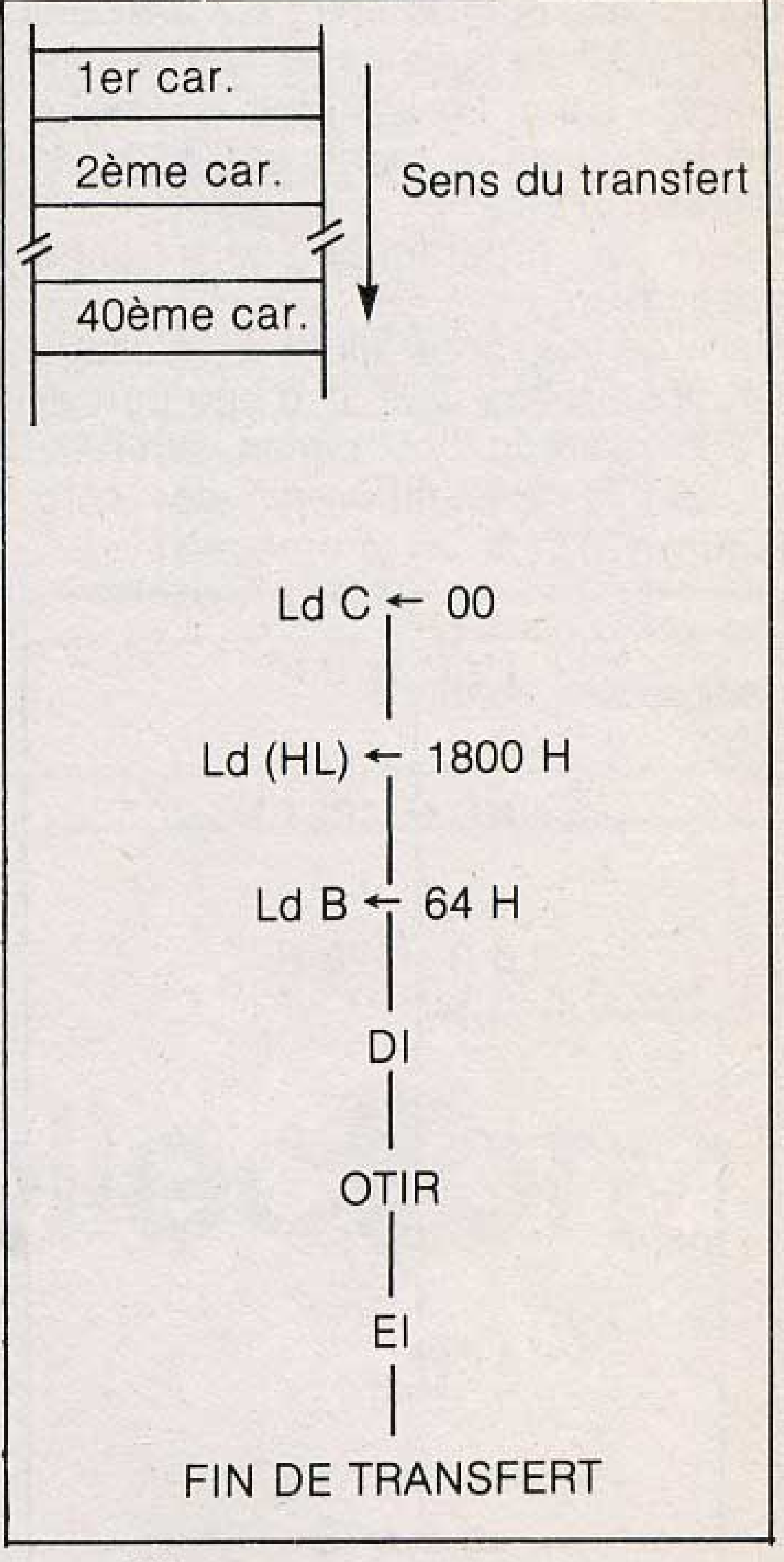


Figure 156

Une seconde instruction analogue à «OTIR», mais dans laquelle **au lieu d'incrémenter (I), la paire de registres HL, celle-ci est décrémentée (D)** est disponible, et se note «OTDR». Le code hexadécimal correspondant est : ED BB.
Dans le premier exemple (figure 133), la boucle ne peut pas être remplacée par l'instruction OTIR. Il existe cependant une instruction similaire **mais sans la fonction saut intégré** : elle se note OUTI, dont le code hexadécimal est : ED A3.
Description :
Le contenu de l'emplacement mémoire pointée par la paire de registres HL est transféré dans le Port de sortie adressé par le contenu du registre C. Le registre B est décrémenté, et le registre HL est incrémenté.
Une seconde instruction analogue à «OUTI», mais dans laquelle la paire de registres HL est décrémentée (D) au lieu d'être incrémentée (I) est disponible. Elle se note «OUT D». Le

code hexadécimal correspondant est ED AB.
Exemple :
Supposons que les 40 caractères qui constituent la ligne à imprimer soient placés dans l'ordre des adresses décroissantes à partir de 1827 H (figure 157).
Ecrire le programme dans cette application.
Dans ce cas, nous allons faire usage de l'instruction OUT D d'une part et d'autre part de l'instruction saut relatif si B est différent de zéro (figure 157).

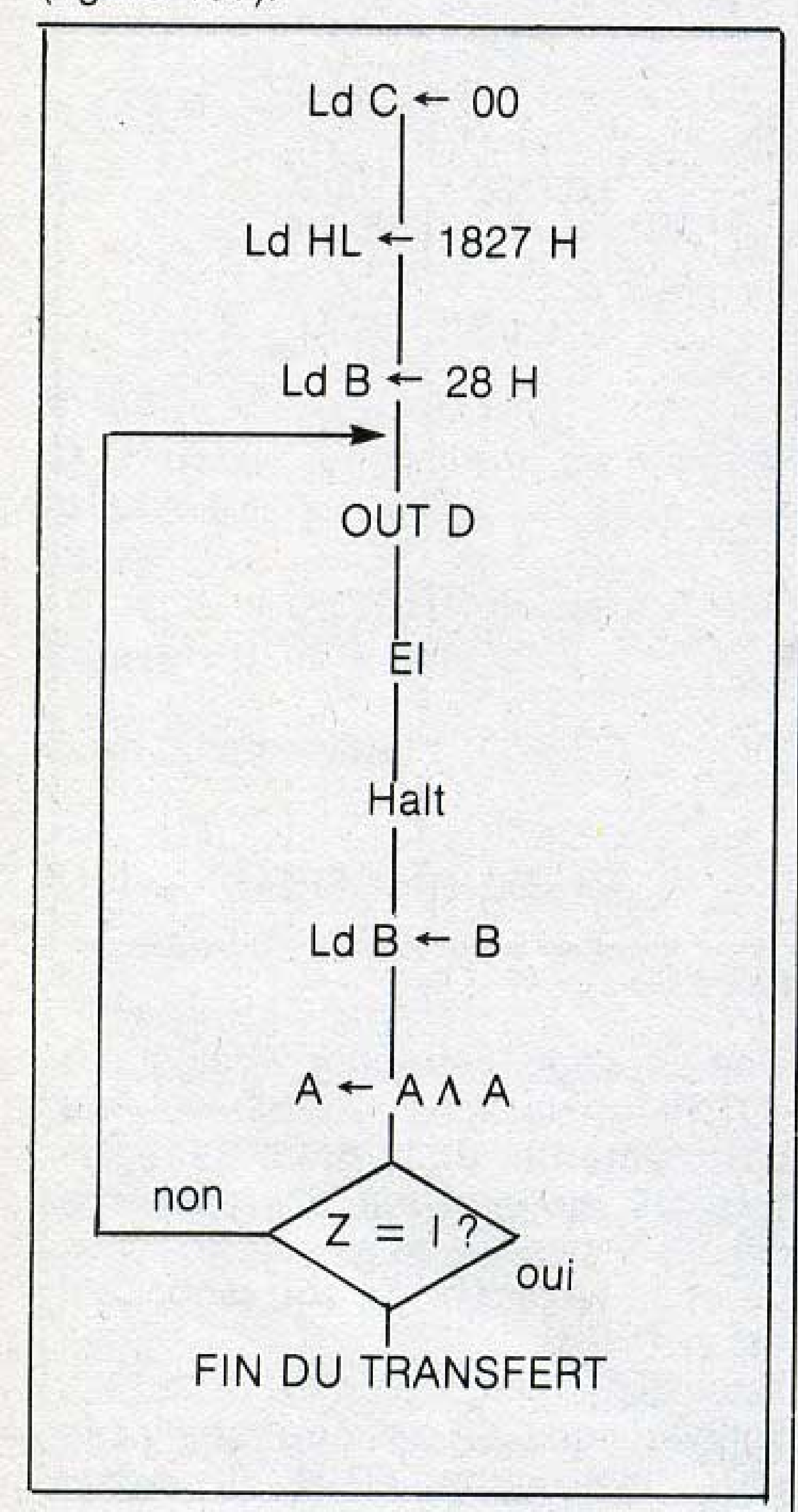


Figure 157

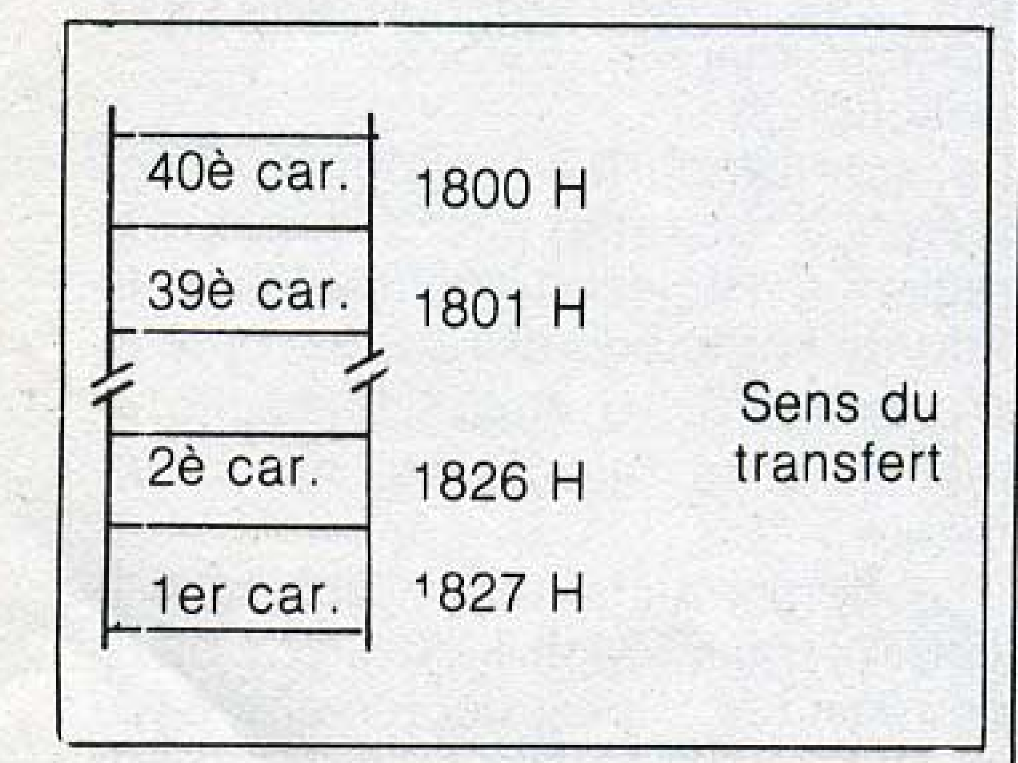


Figure 158

Essayez de trouver d'autres solutions plus simples pour effectuer le test B = 0. En employant par exemple l'instruction DJNZ.

6. Instructions de transfert par BLOC, ENTREE-SORTIE

Nous venons de voir en détail les 4 instructions de transfert par bloc pour la «sortie» d'un paquet d'octets. Il existe 4 autres instructions analogues pour l'«entrée» d'un bloc d'un maximum de 256 octets.
Les deux tableaux suivants résument l'ensemble de ces instructions (figures 159 et 160).

Transfert par Bloc-Entrée

INI	Incrémentation de HL Décrémentation de B	ED A2
INIR	Incrémentation de HL Décrémentation de B Répétée si B # 0	ED B2
IND	Décrémentation de HL Décrémentation de B	ED AA
INDR	Décrémentation de HL Décrémentation de B Répétée si B # 0	ED BA

Exemple
INDR : le port adressé par le registre C est lu et l'octet qu'il contient est transféré dans la case mémoire pointée par HL. Les registres B et HL sont l'un et l'autre décrémentés. Si le contenu de B est différent de 0, l'instruction est répétée.

Transfert par Bloc-Sortie

OUTI	Incrémentation de HL Décrémentation de B	ED A3
OTIR	Incrémentation de HL Décrémentation de B Répétée si B # 0	ED B3
OUTD	Décrémentation de HL Décrémentation de B	ED AB
OTDR	Décrémentation de HL Décrémentation de B Répétée si B # 0	ED BB

Figure 159

7. Instructions de transfert par bloc dans la mémoire

Nous venons d'étudier les 8 instructions de transfert par bloc qui mettent en jeu l'un des ports et un emplacement mémoire.

Nous allons présenter un autre ensemble de 4 instructions bâties sur le même principe, mais qui permettent d'effectuer des transferts de blocs d'octets à l'intérieur même de la mémoire.
La taille du bloc d'octets n'est plus de 256 maximum (soit 1 octet) mais 65 536 (soit 2 octets). La paire de registres BC fait office de compteur, et la détection de 0 s'effectue sur l'ensemble BC.
La paire de registres HL pointe l'emplacement de la source.
La paire de registres DE pointe l'emplacement de la destination.
Avant d'effectuer l'une des quatre instructions suivantes, il est indispensable que le programmeur initialise les 3 paires de registres BC, HL et DE.

LDI : LD (DE) ← (HL)

L'octet contenu dans l'emplacement pointé par la paire de registres HL est transféré dans l'emplacement pointé par la paire de registres DE. Les paires de registres HL et DE sont toutes deux incrémentées. La paire de registres BC est décrémentée.

LDIR : LD (DE) ← (HL)

Répéter jusqu'à BC = 0.
Cette instruction est identique à LDI. La même opération de transfert est effectuée, mais celle-ci est répétée jusqu'à ce que la paire de registres BC soit nulle.

LDD et LDDR

Ces deux instructions sont respectivement identiques à LDI et LDIR, avec la seule différence que les paires de registres HL et DE au lieu d'être incrémentées sont dans ce cas décrémentées.
Le tableau suivant résume les instructions de transfert par bloc à l'intérieur de la mémoire (figure 159).

Transfert par bloc dans la mémoire

LDI	Incrémentation de HL et DE Décrémentation de BC	ED A0
LDIR	Incrémentation de HL et DE Décrémentation de BC Répétée si BC # 0	ED B0
LDD	Décrémentation de HL et DE Décrémentation de BC	ED A8
LDDR	Décrémentation de HL et DE Décrémentation de BC Répétée si BC # 0	ED B8

Figure 160

8. Instructions de recherche de caractères

En plus des instructions de transfert par bloc, le Z 80 possède 4 autres instructions bâties sur le même principe qui réalisent la recherche des caractères.

Le principe est le suivant. Le caractère de «référence» est placé dans l'accumulateur A. Le contenu de l'emplacement mémoire pointé par le contenu de la paire de registres HL est comparé au contenu de l'accumulateur A. (L'opération A – (HL) est réalisée, le résultat n'est pas conservé, A ne change pas).
Deux cas se présentent :

— si la comparaison est vraie (c'est-à-dire A – (HL) = 0) l'indicateur Z du registre F est positionné en 1.
— si la comparaison est fausse (c'est-à-dire A – (HL)#0), l'indicateur Z du registre F est à «0».
Au préalable, la taille du bloc (le nombre de caractères) est placée dans la paire de registres BC. A l'issue de chaque comparaison, le registre BC est décrémenté. Le test suivant est alors effectué. Si le contenu de la paire de registres BC est différent de 0, l'indicateur P/V est positionné à 1. Si le contenu de la paire de registres BC est nul, l'indicateur P/V du registre F est positionné à «0».
Les quatre instructions de recherche de caractères sont résumées dans le tableau suivant (figure 160).

CPI	Incrémentation de HL Décrémentation de BC	ED A1
CPIR	Incrémentation de HL Décrémentation de BC Répétée si BC # 0 ou A # (HL)	ED B1
CPD	Décrémentation de HL Décrémentation de BC	ED A9
CPDR	Décrémentation de HL Décrémentation de BC Répétée si BC # 0 ou A # (HL)	ED B9

Philippe Duquesne